Chapter 1

Detecting Botnets and Unknown Network Attacks in Big Traffic Data

Luis Sacramento - INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
Ibéria Medeiros - Lasige, Faculdade de Ciências, Universidade de Lisboa, Portugal
João Bota - Vodafone Portugal, Portugal
Miguel Correia - INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

CONTENTS

1.1 Introduction

Internet Service Providers (ISPs) provide their customers with communication services that are easily abused by cyber-criminals. The major ISPs are interested in reducing malicious activity and detecting it is often a first step towards that goal. However, these organizations now run backbones with bandwidths in the order of 1 to 10 or even 100 Gbps [1], which support communication services that were not possible in the past, but also make detection a difficult challenge.

A particular pernicious case of network abuse are *botnets*. These networks of compromised hosts (zombies or bots) are platforms for a large set of illegal activities, e.g., for executing distributed denial of service (DDoS) attacks, disseminating malware, running phishing attacks, and supporting click fraud [12]. ISPs have necessarily a role to play in the mitigation of botnets, as they are key traffic intermediates [36]. Therefore, ISPs have to be able to detect them.

ISPs often use *Network Intrusion Detection Systems* (NIDSs) for identifying malicious activity. Classical NIDSs do *Deep Packet Inspection* (DPI), i.e., they analyze 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 behavioral pattern. Classical NIDSs also fall in one of two categories: *signature-based NIDSs* and *behaviour-based NIDSs*. These approaches require, respectively, knowledge about existing attacks (signatures) and traffic without attacks (normal behavior) for training purposes, neither of which is simple to obtain [2]. This kind of analysis is feasible in reasonably slow link connections, but not in modern high-speed backbones. Furthermore, nowadays most of the traffic payload is encrypted due to the adoption of secure protocols such as TLS, HTTPS, SSH, and IPSec, which makes this kind of inspection even harder and less useful [28].

The difficulty of monitoring high speed traffic has led Cisco to introduce the concept of *network flows* in the context of the NetFlow router feature [4]. The concept was later adopted by all major router vendors and standardized by the IETF [5]. A flow can be defined as a sequence of packets with a common set of features, passing through an observation point, in a given interval of time. Flows are a way of monitoring communication in a summarized way, without inspecting the content of the packets, using instead high-level information about connections (source/destination IP address, source/destination port, etc.) but not the data transferred itself. Analyzing this information is more efficient than doing DPI in terms of protection of the privacy of users and consumption of computational resources, once flows do not carry payload content, requiring thus less processing to be analyzed.

Network flows, or simply flows, are an option to the previously-mentioned approaches for network intrusion detection. Network flow analysis allows, e.g., detecting internal and external actions like network misconfiguration and policy violation

[20]. Flows allow detecting many network layer and transport layer attacks. They also allow detecting botnets, because bots perform identifiable network activity such as contacting command and control (C&C) servers or DDoS attacks. Flows generically do not allow detecting application layer attacks such as SQL injection, cross-site scripting, buffer overflows, races, etc.

The use of *Machine Learning* (ML) in the context of NIDS is far from new. A major application of ML is behaviour-based (network) intrusion detection, also called anomaly detection [18, 8]. Both traditional signature-based NIDSs and NIDSs based on flows can also use ML techniques, but the precision and accuracy of these systems depend on the completeness of the knowledge they have about the threats that they will detect, as they need to be fed and trained with that knowledge. ML techniques aim to provide knowledge to such systems, allowing them to discover hidden patterns in input data based on the knowledge they learned, and classify that data. However, even using ML, there are challenges in analyzing network flow data, such as the huge amount of traffic flow becoming from larger and faster networks as, for instance, connection links of ISPs.

This chapter presents a new approach to detect malicious traffic, even if as part of new attacks, and to identify the malicious hosts involved by inspecting network flows. The approach uses a combination of *unsupervised* ML techniques, without *a priori* knowledge, and threat intelligence information to achieve its goal. The approach is based on the following key insights:

- there is much more normal traffic than malicious traffic;
- malicious traffic is qualitatively different from normal traffic;
- and similar traffic within each category (normal, malicious) can be summarized using unsupervised ML.

Out approach involves dividing traffic (flows) in *clusters*. The larger clusters typically correspond to normal traffic, so the smaller clusters are the ones we have to worry about. For the latter, we propose a classification method based on unsupervised ML to classify them as malicious or benign, so detecting malicious traffic and identify malicious hosts. This classification allows reducing drastically the amount of time spent in analyzing the flows, reducing the size of the problem of processing the amount of traffic at the speed of ISP networks.

The approach works in loop, iteratively and continuously detecting network attacks and malicious hosts. Between iterations, clusters are classified and learned, so that this knowledge can be used in the following iterations. This form of learning provides increasing autonomy to the system and may significantly reduce the network managers' constant need for intervention, although not being completely free from human intervention, as no NIDS is. In fact human intervention is unavoidable when the goal is to *detect attacks without requiring either previous knowledge about attacks (signatures) or traffic without attacks (clean traffic for training)*.

The chapter also presents the FLOWHACKER NIDS that implements our approach. This tool uses the Hadoop MapReduce platform [7, 30] to summarize networks flows and a set of ML algorithms to process these summaries, besides pro-

viding visual tools for human analysis. We evaluated the FLOWHACKER NIDS with two kinds of traffic and it identified botnet C&C servers, SSH brute-force attacks, and denial of service events. For validating the system we used a synthetic traffic flow data set [29]. For testing the system we used real data from a ISP, a large Portuguese telecommunications company with a few million customers, which provides Internet, TV-over-IP, phone-over-IP, and GSM/3G/4G cellular phone services. FLOWHACKER was able to detect several cases of botnet activity.

The main contributions of the chapter are: (1) an approach for improving network security based on the inspection of network flows by using a combination of unsupervised ML techniques to detect intrusions; (2) an iterative learning process; (3) a NIDS that implements the approach; (4) an experimental evaluation that shows the ability of the system to detect intrusions in computers communication using network flows.

The remaining of the chapter is organized as follows. Section 1.2 presents background on network flow field and discusses related work. Section 1.3 presents the approach, Sections 1.4 and 1.5 present more details about it, and Section 1.6 its implementation. Section 1.7 presents and discusses the evaluations results, and Section 1.8 concludes the chapter.

1.2 Context and Related Work

This section provides an overview of intrusion detection using network flows. Section 1.2.1 provides some insight on some of the existing tools to perform flow analysis. Section 1.2.2 gives an overview of some of the most addressed network intrusions and respective works/tools that show how to detect them using a flow-level analysis rather than payload inspection.

1.2.1 Network Flows and Basic Flow Tools

The first network protocol to handle *network flows* was NetFlow, developed by Cisco [4]. It consists in a built-in feature in the Cisco routers, and is used to collect and export flow records. Since then, it has been evolved and its recent version – NetFlow v9 – already includes integration with other protocols, such as MPLS (Multiprotocol Label Switching).

Being network flow technology built-in in network devices, it allows to select, from all the traffic passing through that device, the traffic that matches the set of features that were previously defined by the network administrator, in order to obtain what he wants to analyze. For example, by deploying this technology 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.

Apart from NetFlow, many other vendors have their own implementation for flow collection and exporting. Examples of such implementations are NetFlow-lite, sFlow, NetStream. However, due to the heterogeneous nature of these technologies between different vendors, the Internet Engineering Task Force (IETF) created the IPFIX (IP Flow Information eXport) protocol [5] to standardinze flow collection and exportation, allowing thus for the clients to easily deploy their flow-based applications. As previously stated, packets that share common properties are grouped in flows, and in the IPFIX terminology these properties are referred as *flow keys*. They can form, for example, the following tuple: `(IP_source, IP_destination, port_source, port_destination, typeOfService)`.

In order to simplify the collection and extraction of flows, some tools were developed. The *nfdump* tool [23] is one of them. The tool is compatible with versions v5, v7 and v9 of NetFlow and supports data conversion to plain text (in form of *txt* files). *nfdump* reads the NetFlow data, stores it into binary files, and performs some analysis on it, such as some statistics and aggregation.

SiLK (System for Internet-Level Knowledge) tool [31] is a widely-deployed flow analysis tool developed by the CERT Network Situational Awareness Team. Its application is more appropriated for analyzing of traffic on the backbone of a large enterprise or mid-sized ISPs. It is compatible with both IPFIX and NetFlow (versions v5 and v9). Like *nfdump*, it allows to convert NetFlow data to some specific format, and also has built-in tools to analyze these files, such as performing filtering on the gathered flows and retrieve statistical data.

1.2.2 Intrusion Detection based on Network Flows

Several papers presented flow-based intrusion detection schemes for specific network attacks, including botnets [21, 15, 16, 34, 12, 38], and others such as port scans [32, 17], worms [6, 10, 9], and denial of service [25, 19, 13]. Each of these approaches was designed to detect only one of these attack types, but they are related to our work and useful to explain how flows can be used to detect a certain attack.

An increasing trend in intrusion detection systems is the use of ML techniques [20, 33]. ML 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 [20]. This knowledge may be refined and improved at each iteration, by learning from previous experiences and observations. Such methods have been used in several and different applications, in many different fields of science, such as natural language processing (NLP), speech recognition, bioinformatics, spam detection, network intrusion detection, among many others.

ML algorithms can be divided into two major types: supervised learning and unsupervised learning. The first one relies on a labeled training data set. The data set consists in a set data labeled and categorized in classes by humans that aims to train the system, making correspondences between features and their meaning or interpretation that is expected to the system. After the training phase, the system is

ready to classify input data based on the learning that obtained during the training. Examples of supervised algorithms are Decision Trees, Naïve Bayes, and Support Vector Machine (SVM). For the other hand, unsupervised learning does not have trained labeled data set. In contrast, it receives as input a unlabeled feature vector, and then it is processed for discovering similar groups into it. Clustering is an example of this kind of learning and K-Means is the most applied algorithm for clustering.

In the field of network intrusion detection, ML techniques has been able to classify network traffic and identify both anomalous patterns and potentially harmful users [18, 8, 22]. When NIDS systems integrate this technique, the adopted strategy is usually *behavior-based* detection (or anomaly detection), in which normal traffic patterns are differentiated from anomalous ones. It focuses its attention on finding patterns that would not be expected from the user's behavior. Unlike what *signature-based* NIDSs detect, these patterns are unknown to the system, as they are trained with intrusion-free data. However, this approach requires clean traffic for training, i.e., traffic that does not contain attacks, which is difficult to obtain in ISP networks in production.

Portnoy et al. presented a scheme to detect intrusions based on clustering that is neither behavior- nor signature-based [26]. However, that work does not consider the iterative model we do and does not use flows. The Unsupervised Network Intrusion Detection System (UNIDS) was able to detect unknown attacks without requiring any labelling, signatures or training [3]. UNIDS uses various clustering techniques such as sub-space clustering, density-based clustering and evidence accumulation. However, it does not consider the iteration process we do. Goncalves et al. follow an approach closer to ours but inspect logs, not flows [14]. That work and a few others use open source threat intelligence in combination with other techniques [14, 35]. A short version of the present work appeared before [27].

1.3 The FlowHacker Approach

The FLOWHACKER approach does flow processing using unsupervised ML algorithms with the assistance of threat intelligence to detect unknown network attacks. As explained in the introduction, the approach is based on the assumption that most of the traffic is legitimate, so malicious traffic is much less, as well as that malicious traffic is qualitatively different from normal traffic. Taking this into account, the application of the unsupervised ML algorithm allows splitting the malicious traffic from the clean traffic, so that the biggest clusters are those containing normal traffic, whereas the smaller ones are those that may be malicious (although that is not mandatory; there may small clusters of legitimate traffic). These assumptions in combination with the use of flows allow to cover (1) the difficulty of reacting to an unknown pattern when real traffic is analyzed, and (2) the slow processing and analysis of the traffic payload. The first drawback may be countered by using an unsupervised ML algorithm, and the second by performing the analysis at flow-level.

The approach works in loop, improving the knowledge that has been acquired in previous iterations. This allows improving detection performance with time. For

each loop iteration all phases involved in the approach are executed. Therefore, for each set of collected flows, the tool gains insights from them, improves such insights with threat intelligence information, applies an unsupervised algorithm on the improved insights for getting clusters, and then classifies the smaller clusters as being malicious or benign hosts by using a classifier method based on a unsupervised algorithm. Lastly, these new classified clusters are added to the existing knowledge for the unsupervised data set to be used in the next loop iterations. This learning phase between loop iterations allows increasing knowledge gradually with every iteration.

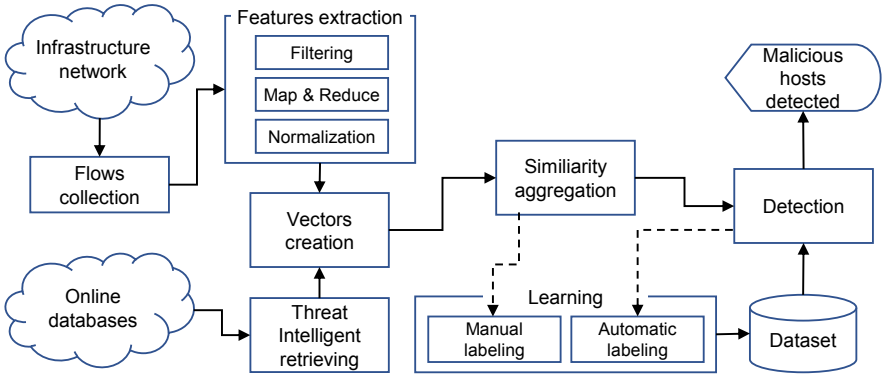

Figure 1.1: Overview of the approach and its six phrases: flows collection, features extraction, threat intelligence retrieving, similarity aggregation, detection, and learning.

The approach comprises six phases, as shown in Fig. 1.1:

1. *Flows collection*: to collect flows from different hosts or routers belonging to a network infrastructure, corresponding to a certain period of time (e.g., a day). Each flow summarizes a set of packets collected from a host during a period of time.

2. *Features extraction*: to extract data from the collected flows in order to create vectors of features that allow characterizing the flows. Flows are filtered to extracting relevant data, afterwards this data goes through MapReduce for getting statistics and summarizing their values, then they are normalized and the vectors created (see Section 1.4.2).

3. *Threat intelligence retrieving*: to automatically retrieve information about threats from online databases, namely blacklists of subnets and IP addresses. Afterwards, this information is used to complement and complete the data of vectors of features (see Section 1.4.3).

4. *Similarity aggregation*: to apply an unsupervised ML algorithm over the feature vectors in order to aggregate similar vectors (vectors with similar feature values), resulting in clusters that represent hosts having a similar behavior. The biggest clusters are assumed to contain clean traffic and the smaller clusters have to be further analyzed (see Section 1.5.1).

5. *Detection*: to detect unknown network attacking hosts. Based on the knowledge acquired in the learning phase a classification method using unsupervised ML classifies the small clusters as being malicious or clean traffic. In the first loop iteration, the classification is done manually, but automatically subsequently when the system has already knowledge from the previous classifications (loop iterations). Clusters are labeled with its classification and the classified data set is updated with them (see Section 1.5.2).

6. *Learning*: to learn the new cluster classifications resulting from the detection phase. For each loop iteration, the clusters classified in the detection phase are learned for later to be used as knowledge in the next loop iterations by the classifier (see Section 1.5.3).

The following sections present the approach in detail.

1.4 Obtaining Vectors of Features

This section presents the process of extracting data from the collected flows and retrieving threat intelligence information from online databases to create the vectors of features with this data. This process constitutes the second and third phases of the presented approach – *extraction features* and *threat intelligence retrieving* (see Fig. 1.1).

As said, a *flow* summarizes a set of packets with similar characteristics that were observed during a period of time. However, a flow summarizes packets coming from different source IPs and going to different destination IPs. Therefore, to obtain information about individual hosts we must aggregate flows by source and destination IPs. These *aggregated flows* summarize traffic sent or received by each host.

To use these aggregated flows in ML algorithms, we represent each of them by a *vector of features*, i.e., a vector of attributes that characterize these aggregates in a way that is useful for our purposes (intrusion detection). The features can be extracted from the aggregated flows directly, or from external sources of threat intelligence data (e.g., the information if a certain IP address appears in a certain blacklist of not).

The next sections present the features we defined to compose vectors of features and these two phases in detail.

1.4.1 Features

The features have to be carefully chosen because the accuracy and precision of the approach depend on that choice. We characterize aggregated flows using 19 features

from 5 categories (Table 1.1): 3 to represent characteristics from *layer 2*, *layer 3*, and *layer 4* of the TCP/IP stack, one for *statistics* about the aggregated flow, and one for threat intelligence data.

Feature		Description	#
L2	AggregationKey	IP address used as identifier (to which the other features relate to)	
	NumSIPs / NumDIPs	The number of IP addresses contacted	1
	LocationCode	Code for the country associated with the address	
L3	NumSports	The number of different source ports contacted	2
	NumDport	The number of different destination ports contacted	3
	ICMPRate	The ratio of ICMP packets, and total number of packets	13
	SynRate	The ratio of packets with a SYN flag and the total number of packets	14
L4	NumHTTP	The number of packets to/from port 80 (HTTP)	4, 8
	NumIRC	The number of packets to/from ports 194 or 6667 (IRC)	5, 9
	NumSMTP	The number of packets to/from port 25 (SMTP)	6
	NumSSH	The number of packets to/from port 22 (SSH)	7, 10
Statistic	TotalNumPkts	The total number of packets exchanged	11
	TotalNumBytes	The overall sum of bytes	15
	PktRate	The ratio of the number of packets sent and its duration	12
	AvgPktSize	The average packet size	16
TI	BadSubnet	This field expresses whether the IP address belongs to a blacklisted subnet	
	MaliciousIP	This field expresses whether the IP address is blacklisted	
	OpenVaultBlacklistedIP	Similar to the above but from another database [24]	
	MaliciousASN	This field signals if the IP address belongs to a blacklisted ASN	

Table 1.1: Set of features that describe an aggregated flow in the form of a vector.

Layer 2, 3, and 4 features

We defined 11 features to represent characteristics from layers 2, 3, and 4, respectively, 3, 4, and 4 features (lines 2 to 12 of Table 1.1).

From layer 2 are extracted the IP addresses, both source and destination. However, it is necessary to identify a vector of features uniquely, therefore, packets coming from different source IPs and different destinations IPs constitute different vectors of features. We defined the `AggregationKey` feature to have this role of identifying a feature vector uniquely, which receives the IP address (source or destination). This choice was inspired by [3] that used source and destination IP addresses to distinguish groups of *1-to-N* and *N-to-1* anomalies. The `NumSIPs/NumDIPs` features represent the number of all different IP addresses contacted contained in the aggregated flow, for that key, whereas the `LocationCode` feature contains the country code of the address IP that fills the `AggregationKey` feature.

Features from layer 3 are related to source and destination ports, the SYN flag from TCP, and ICMP protocol. Features `NumSports` and `NumDports` represent the first two characteristics, containing the number of all different source ports and destination ports contained in the aggregated flow, for that key. If the SYN flag is observed, the `SYNRate` feature will be filled by the number of times that a SYN flag is sent divided by the total number of packets in the aggregated flow for that key. For the `ICMPRate` feature the same procedure is applied, i.e., the number of times the

ICMP protocol is used divided by the total number of bytes of the aggregated flow for that key.

Features from layer 4 are related to the application protocols that are used in the aggregated flow for that key. Features `NumHTTP`, `NumIRC`, `NumSMTP` and `SSH` represent the number of occurrences of contacting ports 80, 194 or 6667, 25 and 22, respectively for the HTTP, IRC, SMTP and SSH protocols.

Statistic features

Four statistic features (lines 13 to 16 of table) were defined to summarize an aggregated flow for a given key, regarding its number of packets and size. `TotalNumPkts` and `TotalNumBytes` features are defined to represent these two characteristics which are obtained by summing all the values of them, i.e., the number of packets and the number of bytes, respectively. From these two features we obtain another two: `PktRate` and `AvgPktSize`. The `PktRate` feature gives the aggregated flow packet rate, which is obtained by dividing `TotalNumPkts` by the total duration of the aggregated flow for a given Key, and the `AvgPktSize` feature gives the aggregated flow packet average, which is obtained by dividing `TotalNumBytes` by `TotalNumPkts`.

Threat intelligence features

The last 4 lines of Table 1.1 present the features related to evidence of threats in the aggregated flow. The *BadSubnet* and *MaliciousIP* features indicate if the aggregation key (`AggregationKey`) belongs to a subnet or malicious IP blacklist, whereas the *OpenVaultBlacklistedIP* and *MaliciousASN* features have the same mean than *MaliciousIP* but the threat intelligence source is different. All of these features are binary, meaning that their content is 0 or 1, i.e., the key does not belong or belongs to such lists.

1.4.2 Flow Feature Extraction

The first 15 features are extracted from the aggregated flows in the *feature extraction* phase. This phase is composed by three tasks, namely, *filtering*, *mapping & reducing*, and *normalization* (see Fig. 1.1) which are described as following.

Filtering

Upon the receiving of the gathered flows, a filtering is performed to get the characteristics related with 9 features referenced above (lines 3, and 5 to 12 of Table 1.1). Since the data of these features are contained in fields of the header protocols from layers 2, 3, and 4, the easiest way of filtering them consists in removing some unnecessary characteristics from the flows (e.g., its payload content and date). Each packet is unencapsulated for accessing to the protocol header fields, extracting data from these fields according to the defined features, creating a tuple with these data, and storing the tuple temporarily to later be processed. Specifically, 9 characteristics are extracted to form a tuple, namely 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. A representation of a tuple is <`srcIP, dstIP, srcPort, dstPort, #pkts, protocol, flag, #bytes, duration`>.

MapReduce

This task processes the tuples, by first aggregating them in order to form aggregated flows, secondly for each aggregated flow to be represented by a vector of 15 features presented in Table 1.1 (lines 2 to 16). Aggregating flows means to merge into a single representation all the tuples that have the same source or destination IP address (destination and source addresses for destination and source aggregations, respectively). Notice that the flows collected from a host are those that outgoing, and so represented by the source IP address, and those that incoming, which are represented by the destination IP address. To get the vector of features it is necessary mapping all data inside a aggregated flow, and then reducing these data, merging it into one.

To achieve such actions, the MapReduce paradigm is used. MapReduce was first introduced by Google [7]. It allows processing big data in parallel in large server clusters. For that purpose, it divide processing jobs in two phases that run respectively a *mapper* and a *reducer* function. The mapper phase involves applying the mapper function to each of the input files and obtain a set of pairs <`key;value`>; the reducer phase runs one or more reducers that receive as input the pairs generated by the mappers, and aggregate key pairs with the same keys, performing some operation on their respective values.

Mapper

Following the algorithm, the mapper for every tuple parses it, gets the 9 characteristics extracted in the filtering task and creates a <`key;value`> pair for each one, being the `key` a string in the format `"S/D,feature,IPaddress"` and the `value` the characteristic value for the feature in question. The key is composed by three elements that combined allow uniquely identify the origin of the feature, which is associated to a source or a destination IP. The `S/D` element denotes if the tuple represents an outgoing (`S`) or an incoming (`D`) flow, being `S` and `D` representatives of source and destination IP address, respectively. The `feature` element is the feature parsed from the tuple, and the `IPaddress` element is the IP address of the sender host.

We defined 22 key pairs, i.e., 11 key pairs for each aggregation key. Fig. 1.2 shows the key pairs for the source IP aggregation key. The last four defined key pairs are regarding the counters for the number of times the protocol port was used, allowing thus calculate the layer 4 features (lines 9 to 12 of Table 1.1). The key pairs to destination IP as aggregation key are analogous, replacing in the key the `S` by `D`, and `srcIP` by `dstIP`.

For example, the outgoing tuple <`192.168.0.105, 10.10.5.2, 80, 80, 52, HTTP, , 1050, 31`> sent by the 192.168.0.105 source IP when received by the Mapper, the following key pairs showed in Fig. 1.3 are produced.

```
<"S,dstIP,srcIP";dstIP>

<"S,srcPort,srcIP";srcPort>

<"S,dstPort,srcIP";dstPort>

<"S,pkts,srcIP";#packets>

<"S,protocol,srcIP";protocol+flag>

<"S,bytes,srcIP";#bytes>

<"S,duration,srcIP";duration>

<"S,HTTPPort,srcIP";yes/no>

<"S,IRCPort,srcIP";yes/no>

<"S,SMTPPort,srcIP";yes/no>

<"S,SSHPort,srcIP";yes/no>
```

Figure 1.2: Key pairs defined for the source IP as aggregation key.

Reducer

The reducer receives the key pairs of every tuple, aggregates them by key, and calculates counts and sums using them in order to obtain the final values needed to fill the vectors of features. When the reducer receives a pair with a key IP address that has not already seen, it creates a new feature vector which the `Aggregation Key` feature is filled with that IP address. On the other hand, when it receives entries with key IP addresses that have already known it aggregates their values in the respective vectors. For example, in order to summarize the number of bytes sent by the 192.168.0.105 source IP, i.e., calculating the `TotalNumBytes` feature, the mapper produces the <`"S,bytes,192.168.0.105";value`> pair for each tuple, being `value` the number of bytes sent in the tuple. The reducer, on the other hand, receives these records, and sum the `value` of all the records that have the same <`"S,bytes,192.168.0.105"`> key, aggregating in this way such values.

After all flows are aggregated in a single one and represented by a vector, the reducer produces the 4 statistic features (lines 13 to 16 in Table 1.1) derived from the other 9 features. Finally, the `LocationCode` feature is fetched by using the `Aggregation Key` value and a IP tracker that references geographic IPs.

At the end of the MapReduce algorithm execution, the tool has two sets of feature vectors: one representing the outgoing aggregated flows (identified by `S`) and another representing the incoming aggregated flows (identified by `D`).

Normalization

Most features carry numeric data, but there is the need to keep every value in one common scale. Moreover, there are some features that are not expressed in a numer-

```
<"S,dstIP,192.168.0.105";10.10.5.2>

<"S,srcPort,192.168.0.105";80>

<"S,dstPort,192.168.0.105";80>

<"S,pkts,192.168.0.105";52>

<"S,protocol,192.168.0.105";HTTP>

<"S,bytes,192.168.0.105";1050>

<"S,duration,192.168.0.105";31>

<"S,HTTPPort,192.168.0.105";yes>

<"S,IRCPort,192.168.0.105";no>

<"S,SMTPPort,192.168.0.105";no>

<"S,SSHPort,192.168.0.105";no>
```

Figure 1.3: Key pairs of an outgoing tuple for the 192.168.0.105 source IP.

ical manner, such as the IP addresses and the country. In these cases, these features are mapped to numerical values, which can be reversed to text. In addition, there are other features (e.g., `NumDport`) that are numeric but their values must be normalized. On the other hand, there are features that do not need to be normalized because they already are, such as threat intelligence features in which their values are binary, i.e., 0 or 1.

Normalizing a set of values means mapping these values to a specific range. We want normalize the feature values to the interval $[0,1]$, where 0 is absolute minimum, and 1 the absolute maximum. To achieve so, given the feature values from a feature vector in the form $X = (x_1, ..., x_n)$, the correspondent normalized $Y = (y_1, ..., y_n)$ vector is obtained using:

$$y_i = \frac{x_i - min(X)}{max(X) - min(X)}, y_i \in [0,1] \tag{1.1}$$

1.4.3 Obtaining Threat Intelligence

In order to complement the information of aggregated flows, the feature vectors are completed with threat intelligence about blacklisted subnets and malicious IPs. This way, online threat intelligence repositories are accessed to retrieve these lists. After flows were processed and feature vector created and filled by MapReduce algorithm, the four threat intelligence features (last 4 lines of Table 1.1) are added to the vectors. Next, for each vector is checked if the `Aggregation Key` value belongs to those lists, and in such case the correspondent features are set to 1, otherwise they are set to 0.

1.5 Detecting Network Attacks

We propose an approach to detect unknown network attacks based on the assumption that the majority of the observed traffic is benign rather than malicious, as well as that malicious traffic is qualitatively different to the regular, normal traffic. To achieve so, the proposed approach uses unsupervised ML algorithms in two fashions: to separate both kinds of traffic, generating clusters, and to confirm if the smaller clusters generated are actually malicious, classifying them.

We propose a detection process using unsupervised techniques over sets of vectors of features (see Section 1.4). An unsupervised clustering algorithm is applied on the feature vectors, aggregating groups of vectors having similar values in their features, forming in this way various large groups of hosts, and some outliers (small groups of hosts). Afterwards, the resulting outliers are classified as being malicious or not by using a unsupervised algorithm, that we call a classifier. According to the mentioned assumption, the outliers may represent an attack, although this may not always be the case. Such outliers could also represent, for instance, some application that are less frequently used by a host, 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 and classify them, in order to differentiate the actual attacks from these benign outlier traffic patterns. This detection process regards to the *similarity aggregation* and *detection* phases of the approach presented in Section 1.3, illustrated in Fig. 1.1, and detailed in next sections.

1.5.1 Similarity Aggregation

The idea behind the unsupervised algorithms we consider – clustering – is to group different instances of a dataset into k distinct groups, i.e., clusters, according to the similarity of their values or characteristics. For instance, applying a clustering algorithm to a dataset of network traffic, it would generate k clusters, where one would be representative of regular DNS traffic, another one would be simple SMTP traffic, and so on. Therefore, we want to apply the same idea of clustering to separate and represent normal and abnormal traffic.

Depending on the algorithm used, the value of k may or may not be chosen automatically. For example, the DBSCAN algorithm [11] does not need a predefined k value, but the K-Means and the Mini Batch K-Means algorithms [37] need it. We opted by algorithms in which k must be specified, since we want to find out which is the best k for dealing with diversity of data. We chosen K-Means and Mini Batch K-Means algorithms because the former is the most used algorithm for that task due to its simplicity and efficiency, and the latter we want to investigate if it behaves as well as the former, and can give us other insights not given by K-Means.

Choosing the number of clusters

Both K-Means and Mini Batch K-Means algorithms require the number k of clusters to be specified. There is no obvious value for k. 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*. 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 total *within-cluster sum of square* (WSS), which is mathematically defined as follows:

$$WSS = \sum_{i=1}^{k} \sum_{x \in c_i} dist(x, c_i)^2 \tag{1.2}$$

Equation (1.2) produces values for k in a specified range, which it is provided by the user. For example, for a specified $k = 30$ and k-means clustering algorithm, the method calculates the WSS for different values of k, by varying k from 1 to 30 clusters. By plotting these values according to the number of clusters k, we obtain a curve that will be decreasing with the increase of k, as we can observe in the plot graph on the top of Fig. 1.4. 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 not provide interesting information at all. Instead, the *Elbow Method* indicates that appropriate number of clusters is when the slope of the *WSS* has a sudden break. Apart from the *WSS*, the *Elbow Method* also calculates the *percentage of variance explained* (PoVE) metric for each value of k (bottom graph of Fig. 1.4) which reflects the ratio of the *between-group variance* (BSS) and *total variance* (TSS), and indicates an optimal k when it suffers an abrupt change.

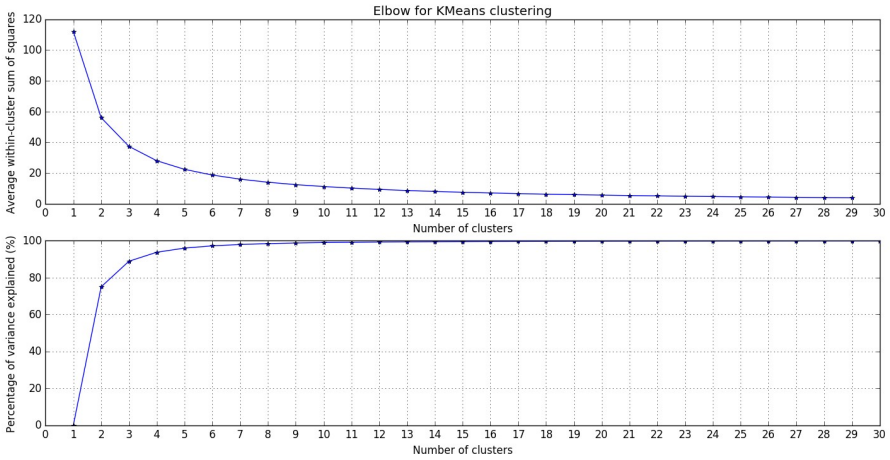

Figure 1.4: Plot of the elbow method.

According to the method and analyzing the Fig. 1.4 as example, the optimal k would be 2 because the biggest slope is found for $k = 2$ (x-axis), which is can not fit for our case. However, there is a rule of thumb often used, which consists in starting off with $k = \sqrt{\frac{n}{2}}$, where n is the number of entries in the data set. Applying this rule we found that $k = 10$ is a number of clusters that successfully and coherently divides the different datasets of various sizes. Observing the Fig. 1.4, we can see that around $k = 10$ the KSS and PoVE values start to stabilize, and its variation is close to 0.

Describing clusters

The potential malicious aggregated flows, i.e., the aggregated flows that can correspond to intrusions, are assumed to be placed in the clusters with smaller size. In order to obtain a coarse grained overview of each cluster's content and to classify each cluster easily, each feature of each cluster is described by its mean value and standard deviation. In this way, it is possible to have an idea of each cluster's behavior and each cluster's feature distribution. Also, this allows describing a cluster's content representing it by a single feature vector composed by the mean values of features and the standard deviation of them.

The resulting vector, we call it *descriptor vector*, and for the classification task we focus on those descriptor vectors that have higher feature values, i.e., higher mean values and standard deviations.

1.5.2 Detection

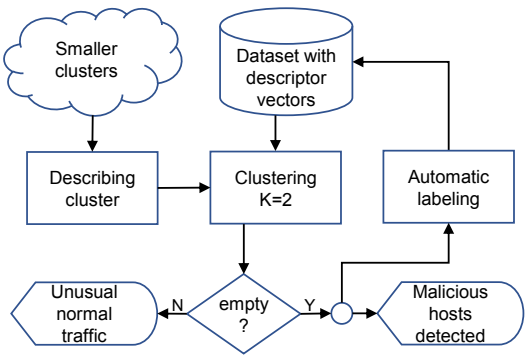

Figure 1.5: Detection phase data flow.

In order to automate the detection of malicious hosts, a classification method based on unsupervised ML is applied to the descriptor vectors resulting from the similarity aggregation phase. This method applies again an unsupervised algorithm

to the descriptor vectors, after they are joined to a temporary dataset comprising descriptor vectors provided from previous loop iterations and classified as being malicious, and then verifies if some outlier results.

The idea behind this classification method relies in the fact that malicious hosts have higher feature values in more than one of their features, and if a clustering technique is applied to a set of instances with these characteristics, there will be a resulting cluster that contains such instances. Therefore, given a set of malicious descriptor vectors and a descriptor vector that we want to classify, if we add this vector to the set and then apply an unsupervised algorithm for $k = 2$, we envisage two possible results: 1) all instances will be put in a single cluster (i.e., resulting an empty cluster), meaning that the descriptor vector is classified as malicious; 2) both clusters will be populated, which one of them is considered an outlier and contains the descriptor vector we want to classify, meaning that such vector is an unusual normal traffic case, and then classified as normal. Figure 1.5 shows the data flow of this classification method and the process explained above.

In this process of classification, we can say that the system runs a unsupervised learning algorithm that is *trained* with the malicious labeled data produced by the previous classifications, and will proceed to classify the traffic that was perceived as being outlier by the first clustering algorithm application (on similarity aggregation phase). 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.

1.5.3 Learning

In a first run of the system, the classification module has not yet any knowledge at all, and so there is a need for a manual intervention that will classify and label the descriptor vectors of the outliers provided from the similarity aggregation phase. Therefore, upon the clustering of the data resulting from the similarity aggregation phase, a manual intervention is performed in order to analyze the characteristics of the traffic – summarized in the form of clusters –, ultimately leading to the production of a labeled dataset that will serve as *training* for the classification method of the detection phase.

This initial manual classification serves as input to the unsupervised algorithm deployed in the detection phase. In the following runs of the system, the dataset will be increasing with the previous classifications made. So, over time, there may not be a need for manual intervention since the classification method will come more and more capable of classifying on its own, which is based on the previous classifications made and learned by the method (see Section 1.5.2). However, we recall that only descriptor vectors classified as malicious are used to compose the dataset used on

clustering task of the detection phase. Therefore, the vectors classified as normal, manually and automatically, are discarded.

1.6 FlowHacker Implementation

To evaluate our approach we implemented it in the FLOWHACKER NIDS, which we developed in Python. The system is composed by two modules – *similarity* and *classification* –, for aggregating the feature vectors in clusters and detecting malicious hosts, respectively. In addition, FLOWHACKER interacts with the Hadoop framework for running MapReduce over the flows, obtaining aggregated flows and vectors of features.

Before using FLOWHACKER, the first step is to gather the flow collection, obtained 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 the SiLK format. Next, a filtering to the flows is performed for getting the nine features needed to compose the feature vectors (see as Section 1.4.2). Afterwards, these filtered features are processed by the Hadoop framework. 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. To support the parallel data processing, Hadoop implements Google's MapReduce algorithm [7]. This model operates on a virtual environment called *HDFS*, which has both *Mapper* and *Reducer* nodes. This model can be divided in two main steps, *Mapping* and *Reducing* that realize the operations described in Section 1.4.2.

FLOWHACKER starts with the similarity module which allows the users managing the clusters, such as change or calculate the number of clusters (k), generate the clusters, and visualize their contents, and then the classification module gets the smaller clusters resulting of the first module, verifies if they are malicious or not, classifying them, and updates the dataset with the malicious ones for further classifications. Moreover, the tool has a terminal interface that was developed to facilitate the similarity aggregation tasks and visualize the outcomes.

1.7 Experimental Evaluation

In order to validate our approach, we evaluated FLOWHACKER experimentally with two datasets: the ISCX synthetic dataset[1] (Section 1.7.1), and real data provided by the large Portuguese ISP (Section 1.7.2).

The objective of the experimental evaluation was to obtain answers for the following questions:

1. Is FLOWHACKER able to detect attacks against synthetic data and real data?

[1] http://www.unb.ca/research/iscx/dataset/iscx-IDS-dataset.html

2. Is FLOWHACKER able to identify the type of attacks performed?
3. How does it perform in terms of false positives and false negatives?
4. Is FLOWHACKER able to detect botnet activity?

1.7.1 Evaluation with Synthetic Data

The ISCX dataset consists in flows corresponding to one week, and aims to provide a complete test dataset for IDSs [29]. The dataset contains attacks of four classes (Table 1.2): probing, to gather information about the network; denial of service (DoS), to compromise system availability; user to root (U2R), in which the attacker leverages access to a normal account to gain root privileges; and remove to login (R2L), in which the attacker exploits some vulnerability to gain access. Moreover, there is a train dataset and a test dataset, whereas the second is a superset of the first, as shown in the table.

Class	Train dataset attacks	Test dataset only attacks
Probing	portsweep, ipsweep, satan, guesspasswd, spy, nmap	snmpguess, saint, mscan, xsnoop
DoS	back, smurf, neptune, land, pod, teardrop, buffer overflow, warezclient, warezmaster	apache2, worm, udpstorm, xterm
R2L	imap, phf, multihop	snmpget, httptunnel, xlock, sendmail, ps
U2R	loadmodule, ftp write, rootkit	sqlattack, mailbomb, processtable, perl

Table 1.2: Attacks in the UNB ISCX train and test datasets (all attacks from the first exist also in the second).

All of the flows in the ISCX dataset are labeled, therefore allowing for a validation of the accuracy of the FLOWHACKER. 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 that was labeled, we proceeded to train our classifier, which is the classifier for further flows to be analyzed.

Cluster Analysis

The dataset is divided in 6 subsets, each one representing a weekday, from Saturday to Thursday (no Friday). Attacks were detected in all of these days, except for Wednesday, that was found to be attack-free. For the data of each day, we did filtering, extraction, and normalization. Next, it was processed by Hadoop, and then we used FLOWHACKER configured for 10 clusters. Next we present the clustering results.

Saturday: By analyzing the contents of the clusters corresponding to this day, we

found one cluster that presented features that are rather alarming. In this one, 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. A study on Brute-Force SSH attacks [17] has 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 indicting the presence of an intrusion), we considered that the flow present in this cluster was performing such an attack, therefore highlighting it as an intrusion flow, as the remainder of the traffic was considered to be normal.

Sunday: The results for Sunday have shown a very different pattern. Unlike the previous day, 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 found 4 that 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.

Monday: On Monday, we found two clusters that immediately distinguish themselves from the rest. The first one has a mean value of 0.998 for the ICMP Rate, being it the cluster with the biggest dimension (it contains 375 different flows); the second one 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 the second cluster, throughout the whole evaluation of the system, there was 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, other four clusters 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. Such pattern may be attributed to a 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. 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.

Tuesday: For Tuesday, we observed that there were multiple clusters with a very high

value for the ICMP Rate. However, this feature appear alone, i.e., it is the only feature in these clusters that has a relevant high value, no other features show up, apart from one cluster that also has a high value for the average packet size, which also does not correspond to a recognizable pattern. 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. Also, it has a high packet rate, average packet size and total number of bytes. From what we have seen so far, this can only correspond to an attack, and therefore the content of this cluster was labeled as being an attack.

Wednesday: As mentioned, no attacks were identified for Wednesday.

Thursday: For Thursday, just like for Saturday, there is one cluster that was 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, three other clusters 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 two clusters were also labeled as malicious.

Unsupervised Classification

In parallel with the daily analysis, FLOWHACKER may also autonomously identify malicious activities using the classification method implemented, which classifies the smaller clusters resulting from the similarity module based on an unsupervised algorithm and a training dataset. Before the FLOWHACKER is able to classify data it is needed, at least, malicious labeled data from the first day, which results from the manual intervention described in Section 1.5.3. 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.

The classifier was trained for the first day with data from the analysis for Saturday, as seen in Section 1.7.1. As expected when the system processed the clusters for that day, it correctly identified the malicious flow. However, when trying to classify the results for Sunday, the classifier did not found any sort of malicious activity. This was due to the fact that the system's only knowledge about the intrusions observed 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 for Sunday, the classifier 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 1.3 shows the results with more detail. Along the system iterations, FLOWHACKER detected 7 attacks, which correspond to flows 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. Although we observed

	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday
True Positive	1	3	2	0	0	1
False Positive	0	1	0	1	5	4
False Negative	0	17	4	3	0	0

Table 1.3: FLOWHACKER results with the synthetic dataset, showing how results improve with the number of iterations.

24 false negatives, it is visible that they were decreased during the iterations, which is justified by increasing of knowledge that the system was learned. Regarding the Wednesday, the system misclassified 5 flows (i.e., false positives) 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. At the end the system misclassified 11 flows as being malicious. This means that it needs some refine in the classification method. However, we prefer to have a system giving false positives than unreported attacks, i.e., with false negatives.

Result Validation

The ISCX dataset is a dataset with known attacks put in a database containing information about them. The dataset is used as the ground truth validation, so we compared the FLOWHACKER results with this ground truth to validate them and find out the accuracy and precision of our system. Each flow belonging to ISCX is identified by a unique ID, meaning that the flows contained in our clusters have this ID. Therefore, through this ID we were able to traceback it to its IP address (which is stored in the database), and this way we were able to identify the malicious hosts.

We were able to verify that all clusters containing malicious flows identified by our system actually such flows are malicious, meaning thus that our system is able to detect and classify correctly attacks under traffic analysis. Also, we observe that correctly we identify the type of attack evolved in such flows. However, we also verified that our system generates some false positives and misses some malicious flows, i.e., has some false negatives, such as is evidenced by results in Table 1.3.

These results suggest a positive answer to questions 1 to 3.

1.7.2 Evaluation with Real Data

The following results were obtained from data provided by the above-mentioned ISP. 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. All the data that reaches the Portuguese company network arises from the ISP, that connects to Portuguese company 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 are needed extra security measures, such as NIDSs. So, the data that reaches these routers from the outside (i.e. the data incoming from the ISP) is collected by NetFlow, for further analysis. For the purpose of this evaluation, a data sample was collected for a few hours. The dataset was obtained without *a priori* knowledge about the existence of attacks, so we had no data to validate our results, unlike what happened with the synthetic dataset (Section 1.7.1).

After performing the filtering and MapReduce phases, we obtained two subdatasets comprising the source flows and the destination flows, respectively. Therefore, the following analysis reports to each of these subdatasets. The clustering was performed with the number of clusters set to 30 (i.e., $k = 30$).

Source aggregation key clustering

In the content of the source aggregation key clustering, we observe five clusters that present high feature values, as shown in Table 1.5 (clusters 13, 15, 17, 21 and 30).

The first 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 threat, 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 the second one, 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 the third cluster, 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.

The fourth presented a high number of IRC (which is used as a portal for botnet's C&C communications) communications, alongside with a high average packet size. This feature distribution led us to consider that this could a botnet communicating, and thus labeling it as malicious hosts. Moreover, we decided to track the IP addresses, and check them against a number of public available blacklists, which confirmed and categorized the IP addresses as sources of Botnets/Spam. One of the IP addresses lead us directly to an authentication page of a C&C.

When analyzing the last alarming cluster, we observed that it presented 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.

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
# Flows	1605	51773	6485	13305	529	1730	1729	21507	8523	8522	1498	4686	10	5653	1	824	5	4606	1864	1676	12	107	13	2233	2264	8091	10	23897	16843	35
Features																														
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-	-	-	0.37	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.67	-	-	-	-	-	-	-	0.38; 0.18	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.24; 0.16
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.54; 0.18	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.63; 0.21	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	-	-	-	-	0.61; 0.21	-	0.84	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.26	-	-	-	-	-	-	-	-	-
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 1.4: Source key clustering content using K-Means.

Destination aggregation key clustering

In relation to the destination aggregation key clustering, we observe that there are 5 clusters (16, 20, 22, 25, 29) with alarming features (Table 1.5).

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 cannot 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 we interpreted them a simple 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 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

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
# Flows	34565	1447	7734	988	312	17180	53767	497	**7**	2442	2177	3089	232	62515	644	**12**	1242	1699	4987	**84**	93	**23**	10883	754	**36**	22543	16	503	**3**	8680
Features																														
1	-	-	-	-	-	-	-	-	**0.84; 0.11**	-	-	-	-	-	-	**0.41; 0.07**	-	-	-	**0.16; 0.07**	-	**0.21; 0.05**	-	-	**0.15; 0.09**	-	-	-	**0.16; 0.09**	-
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**0.22; 0.01**	-	-	-	-	-	-	-	-	**0.24; 0.09**	-
3	-	-	-	-	-	-	-	-	**0.86; 0.16**	-	-	-	-	-	-	**0.59; 0.18**	-	-	-	-	-	**0.52; 0.14**	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**0.84; 0.15**	-
5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.40; 0.18	-	-	-
6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	**0.26; 0.16**	-	-	-	-	-	-	-	-	-	-	-	-	**0.12; 0.11**	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	**0.85; 0.17**	-	-	-	-	-	-	**0.45; 0.11**	-	-	-	-	-	-	-	-	**0.12; 0.05**	-	-	-	**0.10; 0.01**	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	–	-	-	-	-	-	0.23; 0.01	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	**0.26; 0.07**	-	-	-	-	-	-	-	-	-	-	**0.15**	-	**0.21; 0.07**	-	-	**0.45; 0.18**	-	-	-	**0.40; 0.05**	-
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 1.5 Destination key clustering content using K-Means.

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.

Discussion

Table 1.6 summarizes the information regarding the intrusions detected throughout this analysis, emphasizing those related to botnet activity. For the feature numbers, refer to the last column of Table 1.1. The table shows that FLOWHACKER was able to detect and identify attacks in real traffic of a company, both in incoming and outgoing traffic. Such results allow us to verify that real threats are effective and a concern for companies. Also, they allow us to verify that our system is effective in detection of such threats, including botnets, and can be used by companies for avoiding them. This allow us to answer positively to questions 1 to 4.

Cluster #	**Aggregation Key**	**Highlighted Features**	**Type of Attack**
15	Source	1, 3, 5, 8, 11, 15	Spam / DoS
16	Destination	1, 3, 6	DoS
17	Source	10	Brute-force SSH
20	Destination	1, 2, 15	Network scan
21	Source	9, 16	*Botnet communication*
22	Destination	1, 3, 8, 15	Web application probing
27	Source	1, 2, 5, 8, 11, 15	*DDoS IRC botnet*
29	Destination	1, 2, 4, 11, 15	*DDoS botnet*

Table 1.6: Real data analysis results with botnet cases emphasized.

1.8 Conclusion

This chapter presents an approach and a system to analyze traffic from fast networks, such as the fast connection links of Internet Service Providers (ISPs), in which traditional IDSs are limited due their incapacity to analyze big amounts of traffic that circulates in these links and to analyze encrypted data.

The system is based on analysis of network flows, which turns it capable of analyzing such connection links. The approach behind of the system allows detecting malicious hosts without requiring previous knowledge about what we were looking for or clean training data . A combination of data mining techniques for the feature extraction from netflows, and ML techniques for data analysis allows the detection of malicious behaviors without requiring specific training, except for the inevitable human intervention in a first run of the system.

The FLOWHACKER NIDS implements the approach and it was evaluated with both synthetic and real data, being the real data provided by a large Portuguese ISP. The results of both analysis suggest that the system can be used in detection of threats, such as DDoS command by botnet's C&C communications detected in the real data analysis.

Acknowledgements. This work was partially supported by the EC through project H2020-700692 (DiSIEM), and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/00408/2013 (LASIGE) and UID/CEC/50021/2019 (INESC-ID). We warmly thank Henrique Santos for feedback on a previous version of this work.

References

[1] Akamai. State of the internet, 2017. Q1 2017 report.

[2] Guy Bruneau. The history and evolution of intrusion detection, SANS institute, 2001. https://www.sans.org/reading-room/whitepapers/detection/ history-evolution-intrusion-detection-344.

[3] Pedro Casas, Johan Mazel, and Philippe Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[4] B. Claise. Cisco systems netflow services export version 9. RFC 3954, October 2004.

[5] B. Claise, B. Trammell, and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. STD 77, September 2013.

[6] M Patrick Collins and Michael K Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Recent Advances in Intrusion Detection*, pages 276–295. Springer, 2007.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] Herve Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion detection systems. *Computer Networks*, 31(8):805–822, April 1999.

[9] Thomas Dubendorfer and Bernhard Plattner. Host behaviour based early detection of worm outbreaks in internet backbones. In *14th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 166–171, 2005.

[10] Thomas Dubendorfer, Arno Wagner, and Bernhard Plattner. A framework for real-time worm attack detection and backbone monitoring. In *1st IEEE International Workshop on Critical Infrastructure Protection*, 2005.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Data Mining and Knowledge Discovery*, volume 96, pages 226–231, 1996.

[12] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *3rd International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273, 2009.

[13] Yan Gao, Zhichun Li, and Yan Chen. A dos resilient flow-level intrusion detection approach for high-speed networks. In *26th IEEE International Conference on Distributed Computing Systems*, 2006.

[14] D. Gonçalves, J. Bota, and M. Correia. Big data analytics for detecting host misbehavior in large logs. In *Proceedings of IEEE Trustcom*, 2015.

[15] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[16] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium*, 2008.

[17] Laurens Hellemons, Luuk Hendriks, Rick Hofstede, Anna Sperotto, Ramin Sadre, and Aiko Pras. SSHCure: A flow-based SSH intrusion detection system. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 86–97, 2012.

[18] HS Javitz and A Valdes. The SRI IDES statistical anomaly detector. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, 1991.

[19] Aiyung-Sup Kim, Hun-Jeong Kong, Seong-Cheol Hong, Seung-Hwa Chung, and James W Hong. A flow-based method for abnormal network traffic detection. In *IEEE/IFIP Network Operations and Management Symposium*, pages 599–612, 2004.

[20] Bingdong Li, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567–581, 2013.

[21] Carl Livadas, Bob Walsh, David Lapsley, and Tim Strayer. Using machine learning technliques to identify botnet traffic. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, 2006.

[22] Gustavo Nascimento and Miguel Correia. Anomaly-based intrusion detection in software as a service. In *1st International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, pages 19–24, 2011.

[23] NfDump. https://github.com/phaag/nfdump. Accessed: 2018-12-27.

[24] OpenVault. Open vault. http://openvault.com/. Accessed: 2018-12-27.

[25] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, 39(1):3, 2007.

[26] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[27] Luís Sacramento, Ibéria Medeiros, João Bota, and Miguel Correia. Flowhacker: Detecting unknown network attacks in big traffic data using network flows. In *Proceedings of IEEE Trustcom*, 2018.

[28] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *ACM SIGCOMM Computer Communication Review*, 45(4):213–226, 2015.

[29] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012.

[30] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies*, pages 1–10, 2010.

[31] SiLK. https://tools.netsa.cert.org/silk/. Accessed: 2018-12-27.

[32] Anna Sperotto and Aiko Pras. Flow-based intrusion detection. In *12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops*, pages 958–963, 2011.

[33] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of IP flow-based intrusion detection. *Communications Surveys & Tutorials, IEEE*, 12(3):343–356, 2010.

[34] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Botnet detection*, pages 1–24. Springer, 2008.

[35] Ivo Vacas, Ibéria Medeiros, and Nuno Neves. Detecting network threats using OSINT knowledge-based IDS. In *Proceedings of the 14th European Dependable Computing Conference*, pages 128–135, 2018.

[36] Michel Van Eeten, Johannes Bauer, Hadi Asghari, Shirin Tabatabaie, and David Rand. The role of internet service providers in botnet mitigation an empirical analysis based on spam data, 2010. OECD STI Working Paper 2010/5.

[37] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3rd edition, 2011.

[38] Yuanyuan Zeng, Xin Hu, and Kang G Shin. Detection of botnets using combined host-and network-level information. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 291–300, 2010.