Detection of Unknown Network Attackers Through Flow Analysis

Luís Sacramento
luis.sacramento@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2016

Abstract

Network security threats are becoming more and more prominent and the connection links at the Internet Service Providers (ISP) are getting increasingly faster. Traditional Network Intrusion Detection Systems (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 training traffic without attacks. 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 attacks 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 attacks without any *a priori* knowledge.
Keywords: Network Intrusion Detection Systems, Network Flows, Machine Learning

1. Introduction

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 NIDSs 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 passing through an observation point with the same features, 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.

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, namely unsupervised and 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 2. 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 inter-

vention, as no NIDS is.

This approach will allow to detect 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) from where the intrusion took place. 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.

2. Related Work

This section provides an overview of same major contributions in this area. Section 2.1 provides an insight of some of the existing tools to perform flow analysis. Section 2.2 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.

2.1. Network Flows and Basic Flow Tools

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 [21] - the packets' payloads can not be analyzed in real time, for these kinds of speeds. Also, nowadays most packets exchanged have their payload encrypted, making it even more difficult to inspect, even if it was possible to process those many packets in real time. So, what *is* a flow? 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.

In 1996, Cisco developed the first network protocol to handle *network flows*: NetFlow [2]. 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 MPLS that were not supported in the previous versions.

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.

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 this 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) [3]. As previously stated, packets who 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 order to simplify the implementation of these frameworks, there are some tools available. Such is the case of *nfdump* [13], compatible with versions v5 v7 and v9 of NetFlow. This tools allows to read the NetFlow data and store it into binary files, and also to perform some analysis on it, such as some statistical method, aggregation, and it also supports conversion to *txt* files.

Another tool, that is widely deployed, is SiLK - System for Internet-Level Knowledge [19], 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). Such as *nfdump*, it allows to convert NetFlow data to some specific file extension, and also has built-in tools to analyze these files, such as performing filtering on the gathered flows and retrieve statistical data.

2.2. Intrusion Detection based on Network Flows

Nowadays, there are numerous numbers of existing 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 can not 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.

Many studies were considered for this system that approached the detection of Port Scans [20, 10], Worms [4, 7, 6], Denial of Service [15, 11, 9] and botnets [23] with flow-based systems. Each of these approaches was designed exclusively to detect each of these intrusions alone, but were still able to give us insight on what kind of approach to take on the de-

tection of these.

An increasing trend in intrusion detection systems is the use of machine learning techniques [12, 21]. *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 [12]. 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:

1. Supervised learning and 2. 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 users. When it comes to embed this technology in an NIDS, generally the adopted strategy is a *behavior-based* system, 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

A first study was conducted by [16], that was able to design a system that was neither behavior- or signature-based, and allowed to detect intrusions by applying unsupervised learning techniques on the data. Similar to this work, and more recently, a system that goes by the name of UNIDS (Unsupervised Network Intrusion Detection System) was developed [1], 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. Both these studies were shown to have a great performance, and were able to react different types of traffic with great accuracy.

3. System Architecture

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 works aims to cover two major drawbacks of traditional NIDSs:

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 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. Combining these two ideas we are able to design such a system that overcomes these issues.

3.1. Approach Overview

Figure 1 provides an overview of the structure of the system, as it will now be explained, following the workflow.

First of all, upon the receiving of the gathered flows (step 1 of the figure), a filtering will be performed. This filtering will consist in removing some unnecessary features from the flows, e.g. its playload 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, as depicted by step 2 in the figure.

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. It will be taken 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 to the regular, normal traffic. 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

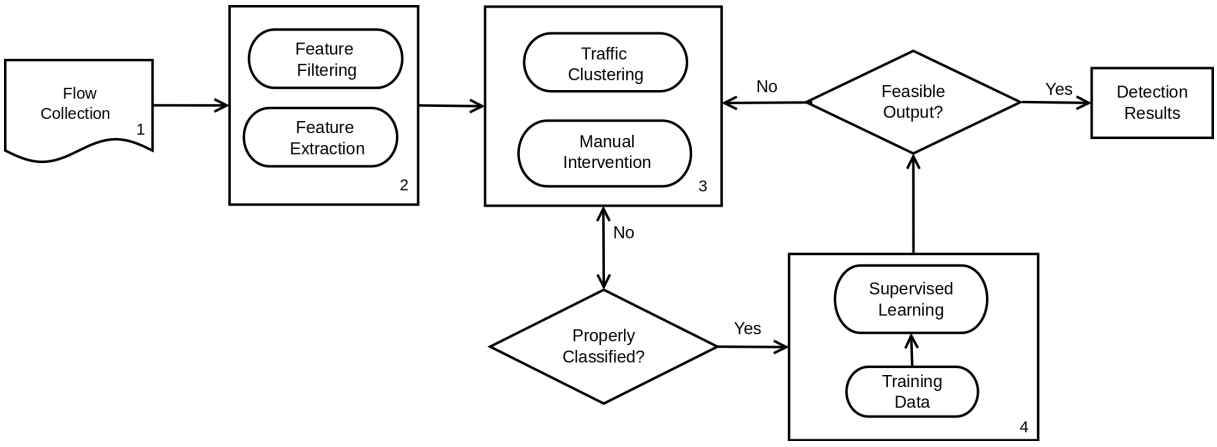

Figure 1: An overview of the intrusion detection approach

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 throughout the study of existing contributions in the intrusion detection community [1, 17, 11] that aggregated flows provide a more precise analysis, e.g. in the case of detecting DDoS. Some attacks, if were to be analyzed using individual flows, would be much harder to detect. The choice of using the source and destination IP addresses as aggregation keys was inspired by [1], 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, altough this may not always be the case. Such outliers could also represent, for instance, some application 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 classify this traffic manually and feed the algorithm once again. The supervised learning algorithm that we will be using a SVM. For the unsupervised learning we will use, as previously mentioned, a clustering algorithm, namely K-Means.

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. All the data that reaches the Vodafone Portugal network arises from the ISP, that connects to Vodafone Portugal 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 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 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 of normalizing the data, so as to remove noise and keep everything in the same scale.

3.3. Feature Extraction

In order to obtain an overall improvement on the system's performance, we decided to do 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 [5], is a Big Data programming paradigm whose 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 user; 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.

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. And this is the part that the Map Reduce algorithm will be applied. The mapper nodes will, for every entry in the dataset, extract its 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 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.

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 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 the reducer receives tuples with an Aggregation Key that already has been seen before, it will generate the remaing features described in Table 1. Features `NumSIPs/NumDIPs`, `NumSports`, `NumDports`,`TotalNumBytes` and `TotalNumPkts` are obtained simply by summing all the values features in each flow, i.e. 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 HTTP, IRC, SMTP and SSH, namely `NumHTTP`, `NumIRC`, `NumSMTP` and `SSH`. 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 compose 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 ICMP protocol is used is divided by the total number of bytes. 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]

3.4. 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 value will belong the interval [0,1], where 0 is absolute minimum, and 1 the absolute maximum. To achieve so, given a dataset entry in the form $X = (x_1, ...x_n)$, the correspondent normalized Y vector is obtained using:

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

3.5. Clustering

As stated in the previous section, machine learning algorithms can be divided roughly in two categories – supervised and unsupervised learning. In this section

[1] Namely: `https://isc.sans.edu/block.txt`, `http://reputation.alienvault.com/reputation.data`, `https://cleantalk.org/blacklists/asn`, `http://www.malwaredomainlist.com/hostslist/ip.txt` and `https://pypi.python.org/pypi/geoip2`

Feature	Description
Aggregation Key	The IP address that will be used as an identifier, to which the below features relate to
NumSIPs / NumDIPs	The number of IP addresses contacted
NumSports	The number of different source ports contacted
NumDport	The number of different destination ports contacted
NumHTTP	The number of packets to/from port 80 (HTTP)
NumIRC	The number of packets to/from ports 194 or 6667 (IRC)
NumSMTP	The number of packets to/from port 25 (SMTP)
NumSSH	The number of packets to/from port 22 (SSH)
TotalNumPkts	The total number of packets exchanged
PktRate	The ratio of the number of packets sent and its duration
ICMPRate	The ratio of ICMP packets, and total number of packets
SynRate	The ratio of packets with a SYN flag and the total number of packets
TotalNumBytes	The overall sum of bytes
AvgPktSize	The average packet size
BadSubnet	This field expresses whether the IP address belongs to a blacklisted subnet
MaliciousIP	This field expresses whether the IP address is blacklisted
OpenVaultBlacklistedIP	Same as the above, but checked from a trusted and well know threat database [14]
MaliciousASN	This field shows if the IP address belongs to a blacklisted ASN
LocationCode	Code for the country associated with the address

Table 1: Features used

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, it 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 of the vector. The previously mentioned data normalization was performed specifically to this step. Depending on the algorithm used, the value of k may or may not be chosen automatically. For example, the DBSCAN algorithm [8] does not need a predefined value for the number of clusters; on the contrary, K-Means requires it. The selected clustering algorithm for the system is K-Means, due to its simplicity and efficiency.

3.6. Supervised 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 preliminar 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.

In order to automate the detection of malicious hosts, a supervised learning module was also introduced. The idea behind supervised learning is to *teach* a machine to produce a certain output for a given input, and this is achieved by feeding the machine with a number of examples - a set of inputs and their respective outputs. 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-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 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 [22, 23].

4. Results & discussion

In order to validate the performance of the system, two datasets were used. First, a dataset provided by the ISCX[2] was used. This dataset consists in flows collected during one week, and aims to provide a complete testbed for IDSs [18]. All of the flows in this

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

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 that has been labeled, we proceeded to train the SVM, which will be the classifier for further flows to be analyzed. Also, an analysis is also performed upon real data provided by the Portuguese ISP Vodafone Portugal.

4.1. Cluster Analysis

First of, 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. 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 attack. For each of these days, data was divided into 10 clusters.

By analyzing Saturday's clustering contents, 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 [10] 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.

The results for Sunday showed a rather different pattern. 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 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.

Moving on to Monday, when looking at their clusters' content, 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.

Analyzing Tuesday, we observed that there were 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, no other features show up, apart from cluster 3 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. Not only it has these features, but it also 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.

Reaching the last day with intrusions, Thursday, just like when we analyzed 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.

4.2. 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 4.1. 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 2 describes this analysis in further detail.

	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 2: Classification Scores

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.

4.3. 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 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 L2R (Local to Remote) or L2L (Local to Local). 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.

The reason that some attack were not identified, it that this system's focus is on volumetric attacks, i.e. attacks that occur in very 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. Vodafone Data Analysis

By looking at the source aggregation key clustering content, we observe five clusters that present high feature values. 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 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 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 thirds, 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 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. 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.

When looking at the destination aggregation key clustering content, we observe that there are 5 clusters whose feature are the most alarming, namely clusters 16, 20, 22, 25 and 29 are those . 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 can't 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 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 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.

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 led to identifying them as being malicious hosts. 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. Also, the system is not scalable for large amounts of data (as in the case of ISPs), as its computational time tends to grow when presented with such volumes of data.

References

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

[2] B. Claise. Cisco systems netflow services export version 9. RFC 3954, RFC Editor, October 2004. `http://www.rfc-editor.org/rfc/rfc3954.txt`.

[3] B. Claise, B. Trammell, and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. STD 77, RFC Editor, September 2013. `http://www.rfc-editor.org/rfc/rfc7011.txt`.

[4] M. P. Collins and M. 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.

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

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

[7] T. Dubendorfer, A. Wagner, and B. Plattner. A framework for real-time worm attack detection and backbone monitoring. In *Critical Infrastructure Protection, First IEEE International Workshop on*, pages 10–pp. IEEE, 2005.

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[9] Y. Gao, Z. Li, and Y. Chen. A dos resilient flow-level intrusion detection approach for high-speed networks. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 39–39. IEEE, 2006.

[10] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras. Sshcure: A flow-based ssh intrusion detection system. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 86–97. Springer, 2012.

[11] A.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong. A flow-based method for abnormal network traffic detection. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, volume 1, pages 599–612. IEEE, 2004.

[12] B. Li, J. Springer, G. Bebis, and M. H. Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567–581, 2013.

[13] NfDump. Nfdump. `http://nfdump.sourceforge.net/`. Accessed: 2015-12-01.

[14] OpenVault. Open vault. `http://openvault.com/`. Accessed: 31/08/2016.

[15] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.

[16] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Citeseer, 2001.

[17] R. Sadre, A. Sperotto, and A. Pras. The effects of ddos attacks on flow monitoring applications. In *2012 IEEE Network Operations and Management Symposium*, pages 269–277. IEEE, 2012.

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

[19] SiLK. Silk. `https://tools.netsa.cert.org/silk/docs.html`. Accessed: 2015-12-01.

[20] A. Sperotto and A. Pras. Flow-based intrusion detection. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 958–963. IEEE, 2011.

[21] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of ip flow-based intrusion detection. *Communications Surveys & Tutorials, IEEE*, 12(3):343–356, 2010.

[22] P. Winter, E. Hermann, and M. Zeilinger. Inductive intrusion detection in flow-based network data using one-class support vector machines. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5. IEEE, 2011.

[23] Y. Zeng, X. Hu, and K. G. Shin. Detection of botnets using combined host-and network-level information. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 291–300. IEEE, 2010.