Hogzilla anomaly-based IDS

Report for the Information Systems Security exam at the Politecnico di Torino

Lorenzo Antonio De Giorgi

September 2020

Contents

1 Introduction

Nowadays we are assisting to a growth in the use of connected devices: home banking services, IoT devices, entire cars systems, all these use Internet connection to offer a more efficent service or to make better the user experience.
Cisco Internet report for 2018-2023 [1] claims that, in the 2023, the 66% of the mondial population will be connected to Internet with 3,6 devices per-capita.
Not just the individual citizen is strictily dependent on Internet connection, but also the critical infrastructures, thus the infrastructures that are vital for fields such as food, agriculture or transportation, have been made more efficient through Internet.

The annual Cyber Threat Report of Austrailian Cyber Security Centre [2] states that the so stricly interdependent systems increase the likelihood and severity of cyber-attacks and the coming of the 5G technology will lead to an even faster transition to the digital economy.
The increase of the number of devices connected, motivates many malicious people to find illegal ways to steal sensitive data, to control devices or also to make them useless by denying services.
Cyber adversaries constantly scanning network services to find vulnerabilities and often they adopt very sophisticated techniques to exploit them.

The growth of the number of attacks and their complexity leads to equally sophsticated couter-measures devising.
For these reasons, nowadays, Intrusion detection system (IDS) is a vital part of the network security since it provides more protection to the network.
A common perplexity is: 'Why install an intrusion detection system if the network is already equipped with firewalls, patched applications and new operating systems? 'The response is simple: because the intrusions still happen.

In the late '70 and '80, intrusion detection was an activity carried out manually, looking at the printed audit logs but the discussed growth of Internet technologies and the complexity of attacks make manually checking impossible. From this, arises the need for a system that automatically scans the logs looking for possible evidences of violations of the protected infrastructures. [3].
Intrusion detection system can reveal or expose the intrusions into a system providing detailed information about them or, for same particular IDS, even react against the attacks with ad-hoc countermeasures.
Developing an accurate intrusion detection is not a simple problem: until a few years ago, IDS works with a centralized architecture, confronting the attacks signatures with a database of known attacks. This mechanism often fails in more complex situations where the zero-days attacks are more frequent and large infrastructures require more scalable solutions.
Developing an intelligent and accurate IDS is the subject of much research which try to integrate the latest in machine learning to the intrusion detection problem.

This work provides an overview of the different IDS technologies 2, considering also open issues and evaluation metrics used for evaluating the performance of intrusion detection systems.
The 3 is exclusively focused on a particular kind of IDS called anomaly-based, and discuss about the idea behind them, the algorithms used and the available tools.
4 treats the main topic of this work, so it presents an overview of the open-source AIDS, Hogzilla IDS, exploring the main components and the general workflow.
5 explained in more details the algorithms and the workflow, that will be tested in the 6 against benchmark datasets.

2 Intrusion Detection Systems

Despite the current urgency to have more defenses against these new attacks, the first generation of IDS dates back to 1980. It is possibile to identify two main steps at the origin of the IDS: the report published by Anderson and the work of Denning.
Anderson published a work in 1980 explaining manners to improve computer security auditing and surveillance by preventing insider attacks through system auditing [5], whereas Denning [6] outlines a framework which then will inspire many researches and projects.
In particular Denning, assuming that an intrusive activity is different from an usual one, explained that the main job of an IDS is to discover an appropriate model for normal activities, in order to easily differentiate it from those malicious ones. He applied these concepts on the first real-time intrusion detection system called Intrusion Detection Expert System (IDES) [17].

The increased spread of malwares and their complexity pointed out in the Introduction, leads to the need to develop efficient IDS to detect them since IDS provide another layer of defense to security techniques. [8]
It is important to understand the principles and the mechanisms behind this technology in order to choose an efficient IDS for a particular purpose. In fact, there are a lot of systems used for intrusion detection which use different methodologies and approaches to discover intrusions on different levels [7].

2.1 Main components

Although the variety of systems, it is possible to identify main components that, though with different implementations, are common to many IDS.
Therefore, based on the analysis of [9], it is possibile to classify the components of IDS in the following:

- *Sensors*: they are in charge of collecting data from the monitored system.
- *Detector (Intrusion Detection analysis engine)*: it processes the data gathered using informations provided by knowledge base and the configuration device.
- *Knowledge base (database)*: it containes the informations necessary to identify threats. Based on the type of IDS, these informations could be very differents: for instance: data profiles, statistics, attacks signatures.
- *Configuration device*: it provides informations about the state of the IDS.
- *Response component*: since on IDS can be an active or a passive component, this component handles the response of IDS against possible threats.

2.2 Architecture

It is possibile to make a first distinction among IDS based on their architectures: there are two main architectures used in IDS, centralized and distributed.
The centralized IDS perform analysis of data on a fixed number of locations (independently of how many hosts are being monitored); distributed IDS perform analysis of data on a number of locations that is proportional to the number of hosts that are being monitored.

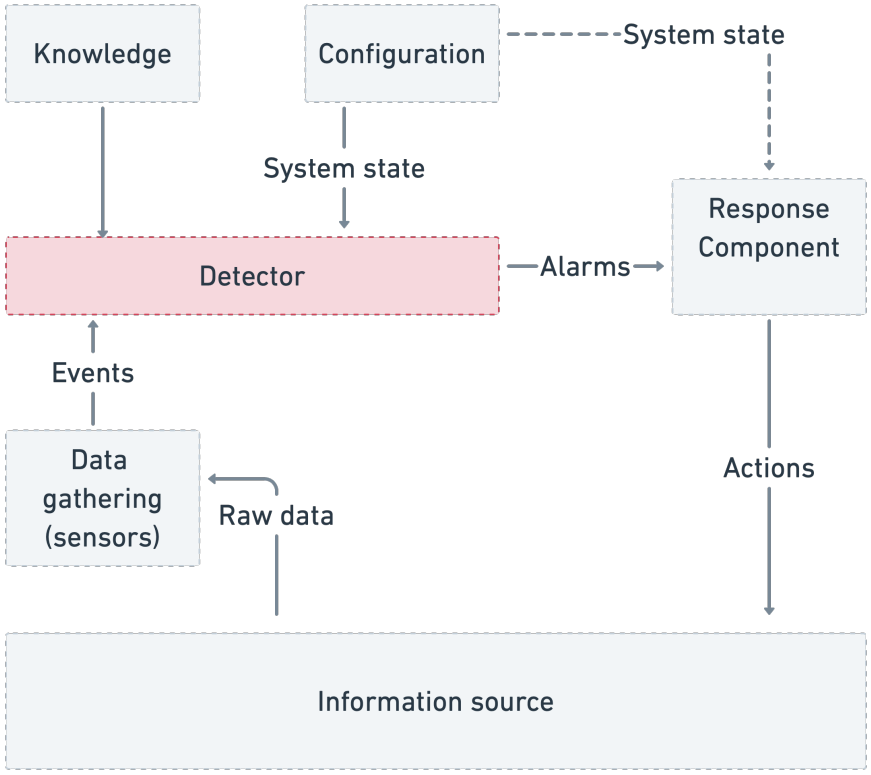


Figure 1: Main components of IDS [9]

Both the approaches can be considered valuable and the design of an architecture strongly depend on the system it is needed to protect. The most evident advantage of a distributed architecture is the scalability: on this kind of IDS it is possible to easily scale to a large number of hosts by adding new components, whereas the size of a centralized architecture is fixed.
Another point in favor of distributed IDS is the graceful degradation of service: distributed architecture is not a single point of failure system, if one component stop running, the rest of IDS can continue working.
Limitations of distributed IDS are the same of any other distributed system such as to guarantee fault tolerance: having data distributed, make difficult to guarantee it compared to a system where the state is centrally stored.
A well detailed comparision between the two architectures can be found in the paper by Spafford and Zamboni [10], in which they also point out the increasing trend to quit centralized IDS in favor of distributed ones.

Given the increasing trend it is important to briefly investigate at least one of the main approach to distributed architecture. An approach in the distributed architecture is using the autonomous agents which can be defined as [10]: a software agent that performs a certain security monitoring function at an host. It is autonomous because runs independently from each other components, so its execution is scheduled only by the operating system and not by another process.
Using mobile autonomous agents has different advantages such as they can be added, removed and reconfigured without altering other components, they can exchange information that could lead to a more complex results. In the previously cited paper, Spafford and Zamboni proposed their architecture based on autonomous agents for building an IDS where they are able to collect and analyze data independently.

2.3 Classification of IDS and their implementations

There are different way to classify IDS using various parameters but there is no an universally accepted classification [9], [13], [14].
In these papers, IDS will be broadly classify based on data source and detection method, where for data source is intended as the information on which the analysis is carried out (gathered by the sensor) and for detection method it is intended the way the detection engine work.
After clarifying the classification, it will be shown some pratical implementations used for detection, from standard techniques to the new applications have emerged in recent years.

There are different way to classify IDS using various parameters but there is no an universally accepted classification.
In this paper, IDS will be broadly classify based on data source and detection method, where for data source is intended as the information on which the analysis is carried out (gathered by the sensor) and for detection method it is intended the way the detection engine work.
After clarifying the classification, it will be shown some pratical implementations used for detection, from standard techniques to the new applications have emerged in recent years.

2.3.1 Data source (HIDS/NIDS)

Data gathered assumes an important role in the effectiveness of detection and they are also decisive for the type of attacks to be detected.
At first, it is possible to distinguish between two technologies: host-based IDS (HIDS) and network-based IDS (NIDS).
HIDS inspect data that originates from the host system and audit sources such as logs, operating system, application system audits. HIDS has the advantages to work with very informative data but the drawback is that those data can be easily modify by successful attacks. Moreover HIDS, analyzing high-detailed data, can impact the performance of the host on which it is running. NIDS extract information from the network traffic. The majority of NIDS extract low-level information such as the IP address of the sender and the receiver, the number of byte transferred, but some other IDS can also perform a more challenging protocol-level analysis.
Despite the different application fields, NIDS has numerous advantages over HIDS given by the diversity of data analyzed, such as the possibility to recognize attacks without being installed on each host and the TCP/IP standardization that makes easier to collect and analyze data provided by different devices.
On the other hand, NIDS require specific hardware, encrypted packets could make difficult the detection and high-speed network could be lead IDS to lost a significant numbers of packets. Some of these disandvantage will be dealt in more detailes later .

2.3.2 Detection methods (SIDS/AIDS))

Depending on the analysis carried out by the IDS, it is possible to categorize them in signature-based IDS or anomaly-based IDS.
A signature-based IDS (also called misuse-based) compares the information gathered with known attacks or signatures; so it strongly depends on the presence of a database which contains updated signatures.
An anomaly-based IDS defines a profile of the system typical behavior and detect the behaviors that are not compliant with the standard profile. [11]

As pointed out by [4] the underlying difference between the two approaches is the same difference between an attack and an anomaly: an attack is 'a sequence of operations that put the security of a system at risk', instead an anomaly is a 'an event that is suspicious from the perspective of security'. A more detailed comparision among the two approches is in the 3.4, but it is possible to briefly sum up them.
The main advantage of the signature-based IDS is its high accuracy in detection of already known attacks [15] but it is ineffective with zero-days attacks or also variants of already known attacks.
On the other hand, an anomaly-based IDS can detect zero-day attacks, because they likely lead to an anomaly of the system.
With the raising rate of zero-day attacks and polymorphic variants of malwares, AIDS are proving to be a potential solution against them [8].

2.3.3 SIDS and AIDS implementations

From the first IDS, on 1980, several solutions have been developed to make detection more efficient both for signature-based and anomaly-based IDS.
Some approches used in signature-based IDS are:

- *Signature based*: in these IDS, monitored events are compared against a database containing signatures of attacks. This approach are not able to recognize new type of attacks since the database have to be continuously updated. A well know signature-based IDS is SNORT [97].
- *Rule based*: in these approach, the attacks are converted in a set of 'if-then' rules. This approach was often used in early stages of IDS because rules represente a regular method for describing security expert knowledge. Some example of rule based IDS was IDES [17] and NIDX [18].
- *State transition tables*: it is based on the construction of a finite state machine where each state represents a state of the IDS and, each transition that cause a changing in the state, represents an event. When a state flagged as malicious is reached, an alert is generated. This technology was proposed in USTAT [19] and in NetSTAT [20].

Some approches used in anomaly-based IDS are:

- *Statistical based*: in this approach, information captured are used to determinate a profile representing the stochastic behaviour of the system. The profile can be constitute by threshold metrics, time serie models, Markov models or other statistical metrics. If the profile generated during the detection process is irregular compared with the normal profile, an event is generated [21].
- *Knowledge-based techniques*: this approach is based on creating a knowledge base that describes the normal profile; each action that deviate from that legitimate profile is considered an intrusion. In this particular approach, the normal profile is usually created based on human knowledge. The main advantage is the reduction of false positive but it requires regular updates of the knowledge base.
- *Machine learning*: this approach uses different techiques such as clustering, decision tree, neural network to extract knowledge from training data. This approach generally is used to improve accuracy and decrease the needed of human knowledge [8].

2.4 Evaluation metrics

There are many classification metrics for IDS, in the following the most used are described and thus the metrics that are going to be used in the experimental part.
The job of an IDS can be interpreted as a classification work, in particular as a binary classification where the input can be classified into one and only one of two non-overlapping classes. In the scope of IDS the input can be classify as "malicious" or "benign".
The computing of the metrics starts with the evaluation of the number of correcly recognized class examples (true positive), the number of correctly recognized examples that do not belong to the class (true negatives), and examples that either were incorrectly assigned to the class (false positives) or that were not recognized as class examples (false negatives). [34]
These values can be represented in the form of Tab. 1 called confusion matrix.

data class	*classified pos*	*classified neg*
pos	True positive (tp)	False negative (fn)
neg	False positive (fp)	True negative (tn)

Table 1: Confusion matrix.

From the values of TP, FP, FN, TN can be calculated the following metrics:

- *Precision*: the number of correctly classified positive examples (TP) divided by the number of examples labeled as positive ($TP+FP$). The range of precision metrics is from 0.0 to 1.0: a value of 1.0 for a general class C means that every example labeled as belonging to the class C, really belongs to it.

$$Precision = \frac{TP}{TP + FP}$$

- *Recall*: the number of correctly classified examples (TP) divided by the number of positive examples in the data ($TP + FN$). The range of recall metrics is from 0.0 to 1.0: a value of 1.0 for a general class C means that every examples belong to the class C has been correctly labeled.

$$Recall = \frac{TP}{TP + FN}$$

Another useful metrics is the F-score, that it the harmonic mean of precision and recall.
F-score maintains a balancing between the precision and recall: if the precision is low, the F-score is low and if the recall is low, again your F-score score is low.

$$Fscore = 2\frac{RECALL * PRECISION}{RECALL + PRECISION}$$

The drawback is that these metrics do not give any information about the examples not correctly classified. For this reason other metrics are introduced such as the *accuracy* that convey the overall effectiveness of the classifier [35]:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

2.5 Issues and proposed solutions

During the years, the approach to IDS is changed: signature-based IDS introduce a wide variety of machine learning algorithms in order to solve inherent and characteristic problems. On the other side, anomaly-based IDS use extensively machine learning, data mining or statistical tecniques to approach in a different way the intrusion detection problem.
Despite this, open issues to solve remains, common to both types of IDS.

2.5.1 Effectiveness

In order to be considered effective, an IDS must have an high detection rate and low false positives([27]): the challenge is to obtain a system that detects close to 100 percent of attacks with minimal false positives.
As discussed previously, the detection rate of signature-based IDS it is affected by inability to detect zero-day attacks or variants of already known attacks, while an high false positive rate affects anomaly-based IDS. Many researches advocate using hybrid IDS that include both anomaly and misuse detection techniques; the hybrid approach used in [28] increases the detection rate from 27 to 146 attacks out of 201 while the combined statistical and signature approach used in [29] increases the detection rate from 77 to 149 attacks out of 180.[26].
An approach to reduce the amount of false positives is to work on the alerts generated by the IDS enhancing the alerts processing technique. The aim is to reduce the noise given by a large amount of false alerts in order to make the process of discovering real threats easier [55].

2.5.2 Performance

Closely connected with effectiveness is the performance understoods as the ability to handle and process a lot of input information without lowering accuracy.
In particular, referring to NIDS, the continuous increasing of network speed could affect to capability of IDS to process all the packets having as consequence to drop packets and not properly detect malicious activities.
A study [36] demonstrates that the open-source IDS Snort, is unable to process packets in high-speed network, dropping packets and decreasing its accuracy.
Different researches demostrates that there is a strict correlaction between handling information and hardware configurations; therefore, one aspect to consider evaluating performance is also the resources usage of the IDS. [37] comparing the speed, memory requirements and accuracy of the open-source IDS Suricata with Snort, on three different configurations. It comes to the conclusion that Suricata can handle larger volume of traffic compared with Snort with similar accuracy but doubling the amount of CPU and RAM used.
The researches, concerning the improvements of these aspects of IDS, claim that integrating machine learning algorithms such as Support Vector Machines, Decision Trees, And Naive Bayes could improve both the accuracy and the false positive rate [38].

2.5.3 Human intervention

Since IDS, expecially AIDS generate many false positive alerts, it is necessary the presence of experts that could verify the danger of all the alerts. This aspect has important consequences for different reasons such as increasing time of detection and so slowing down the possible countermeasures, and make the intrusion detection a more expensive activity.
It is also possible to consider updating database as human intervention: since, as previously

discussed, for the most IDS, expecially for signature-based IDS, it is necessary to constantly update database containing signatures or rules. Also for this issue the underlying problem is the amount of false positives that could not permit to completely make IDS independent from humans. The solution proposed remains the same: using machine learning and hybrid approach to decrease false alarm rate [7].

2.5.4 Evasion techniques

Another challenges for IDS is to overcome the evolving evasion techniques, which are a set of mechanisms that may evade IDS exploiting system implementations. According to [22], it is possibile to group evasion techniques into five categories:

- *Denial-of-service*: this attack aim to run out the system resources or the network bandwidth of the IDS generating a large volume of network traffic. The example in [22] shows how it is possible to slow down Snort manipulating the input traffic and exploiting the worst-case execution of the detection engine. The countermeasures are increasing the availability bandwidth of the IDS and using IDS clustering to enhance the robustness of the IDS; in this way if DoS attacks happen, the administrator can block it and investigate the attack vector.
- *Packet splitting*: it refers to chops IP datagram or TCP stream into very small fragments. If the IDS does not restore the original message, the attack addressed to the victim may be neglect. Some research works [23] and citecountingbloom proposed to split the signatures into short ones and matching them in the split segments.
- *Duplicate insertion*: it is a technique in which the attacker duplicate fragments or segments. He relies on the fact that IDS will handle them in inconsistent way, due to the IDS lacks of information such as network topology or victim operating system. A solution could be a flow modification by picking one interpretation of the protocols and normalizing the traffic according to that interpretation.
- *Payload mutation*: it means that the attacker transforms the malicious payload into a different payload that is still malicious. This aim to obtain a different signature for the attack that may be not on the IDS database. The IDS can decode the payload back but since the victim application could support several encoding methods, the interpretation of IDS have to be consistent with that of the victim.
- *Shellcode mutation*: this technique aims to encode a shell code into polymorphic form to evade the IDS. Encoding techniques could be encryption or compression. [24] proposed a neural network to classify the disassembled instructions in order to find out possible execution paths from the payload.

Intrusion detection techniques are continuously evolving in order to obtain more and more accuracy and a low rate of false positive. Despite the new techniques, some problems could affect both anomaly and signature based IDS; it will be sum up the issue proposed by [7] with some proposed solutions

3 Anomaly-based IDS

An anomaly-based IDS rely on the system behavior to detect an intrusion: broadly, if an activity is not within the system behavior, it is classified as 'abnormal' and therefore as an

intrusion [21]. AIDS are implemented using severel approaches, in general terms, all of them consists of the following modules that is within the detection engine of Fig. 1 [39]:

- *Parameterization*: the instances of the target system are represented in a pre-established form.
- *Training stage*: the model of the normal behaviour system is built in this stage, so the normal and abnormal behaviour are defined.
- *Detection stage*: the observed input information are compared with the model. If the deviation exceeds a certain threshold, an alert is triggered.

As discussed briefly in the first chapter, AIDS are the best solution to detect zero-day attacks but they suffer of high false positive rate, so the experts aims to build AIDS with low false alarm rate and high detection rate exploring different statistical or machine-learning techniques with a fast and appropriate feature selction method.

In 3.1.3, will be presented a classification of AIDS based on the input sources, the different benchmarks datasets and some operations often performed on them.
In the 5.1 will be described the different types of IDS basing on the algorithms used for behavior model building with their advantages and downsides.
A briefly presentation of the currently most used AIDS is given in 3.3, describing the type of IDS.
Lastly, in the 3.4 alla sezione will be presented a more in-depth comparision between signature-based IDS and anomaly-based IDS.

3.1 Input data

Collecting reliable data from the numerous data sources is a complex issue since it is often a trade-off between overheading and effectiveness: for instance logs from routers and firewall could provide simple information such as network connection openings and closing or more complex one such as the entire records of every packet. A system that process high-detailed information could be more accurate, but it performance could quickly degrade.
Collecting the right information, those that are really useful for the analysis, and balancing the trade-off is still an open issue. (Intrusion detection: a brief history).
Preprocessing data could be an effective solution since it is possible to manipulate input data, deleting unnecessary information or even make new ones.

3.1.1 Data preprocessing

Preprocessing includes different phases but all these aim to filter and transform data in a more suitable input for the algorithms. Data preprocessing is an important step taken mostly before applying machine learning or data mining algorithms: [40] claims that data preprocessing stage takes 50% of overall process effort, while the data mining tasks take from 10% to 20%.
Data preprocessing steps include usually: dataset creation, data cleaning, integration, features construction, features selection and reduction.
[55] indicates the following as the main steps for IDS:

- *Dataset creation*: this step aims to choose representative data for testing and training. The dataset should be labeled indicating whether the sample is normal or anomalous. Usually researchers use benchmark datasets to evaluate their results but often these datasets present issues: [27] exposes the major ones. Some important issues could be the privacy problem that not permit to share the whole dataset if the data contained are extracted from real scenarios or the difficult, due to the number of parameters required, to simulate a real-life scenario. [42] proposes some rules to respect so that a dataset can be considered valid. In the 3.1.2 will be discussed in more details datasets and their issues.

- *Feature construction*: it aims to create features that are more effective for the algorithms. In more technical way, it is about the projection of the original dataset into a new space where the linear dependence of features is minimized. Features could be create manually or by using data mining [43].

- *Reduction*: this step consists of discarding useless or redundant features. The main feature selection techniques are embraced within three different groups: wrappers, filters and hybrids or/and embedded. Each of them can be choosed considering the computational cost and performance trade-off: for instance wrapper method is very reliable but it has an high computation cost whilst the filter method is fast and straight-forward but could have bad performance.

By these steps, the original data are transformed in 'observations' that could be represent as a features vector. Moreover, these observations could be labelled with its class.

3.1.2 Datasets

As mentioned in the previous section, some public datasets are used as benchmark to evaluate the performance of IDS [27].
The most famous datasets are:

- *DARPA / KDD Cup99*: it is the earliest effort to create a dataset to test IDS and it was made by DARPA (Defence Advanced Reasearch Project Agency) in the MIT Lincoln Labs. The test data has around 2 milion connection records, each of which has 41 features and is labeled as normal or abnormal. The 1998 DARPA dataset was then used as basis to compose the KDD Cup99 dataset. Though this dataset is very out-of-date (and so it does not contain most recent attacks) and it is widely criticized [44], it is one of the main dataset still used for benchmarking activities [30].

- *CAIDA*: this dataset was created in 2007 contains DDoS attacks. Although widely used, it present two main issues: first it does not contain a diversity of attacks and second, the data does not contain features from the whole network [31].

- *ISCX 2012*: this dataset contains both normal and abnormal traffic from real HTTP, SMTP, IMAP, POP3 and FTP protocols. In the experimental part, this dataset is used due to the diversity of attacks and the real network traffic traces [32].

- *CICIDS 2017*: this dataset present a large variety of attacks such as brute force FTP and SSH, DoS/DDos, botnet and web attacks. Moreover this dataset is composed by both benign and malicious traffic and present a complete topology. Also this dataset is used in the experimental part [33].

3.1.3 Traffic features

Different IDS can use features provided by different sources; in particular, regarding NIDS, that features can be extracted by the packet header, by the analysis of protocol used or by the content of the packets.
Based on this, it is possibile to make another distinction among NIDS in addition to those previously explained in 2.3.

Packet header anomaly detector Packet header anomaly detector (PHAD) is a type of NIDS that attempt to detect intrusion using the values contained in the fields of packet headers. The values can be retrieved from headers of different levels such as data link (Ethernet), network (IP) and transport (TCP/UDP/ICMP). One of the first attempt of PHAD was SPADE [45], a Snort preprocessor plugin that uses the basic features extracted from protocol header to build a normal traffic distribution model for the monitored network: then comparing the observed traffic with the model and calculating an anomaly score, it alerts on possible attacks. [47] claims that using the fields of packet header leads to an inaccurate classifier since most of the fields does not contain traces of attacks; for this reasons, usually PHAD uses clustering methods that ensures that useen but legitimate values are less likely to be reported. SPADE avoids this problem using a very small subset of packet headers, instead [46] , that uses headers of the MAC layer frame to detect attacks against wireless network, applies features selection to find the most releveant set.

Extracting information from every single packet could leads to high computational cost or high resource usage, so it is possibile also to derive features from network flows. Flows is a unidirectional sequence of packets sharing a common key such as the same source address and port and the same destination address and port. From a flow are usually extracted time-based measures such as the number of packets, the duration of flows or the average inter-packet arrival time.

It is also possibile to extract information from multiple flows in a certain time window. In this case, the anomalies detected are unusual patterns of traffic, such as DoS attacks. [48] processes 10 minutes of batch flows on which it calculates features using protocol, union of TCP flags and number of packets: for instance it computes the 'count-dest', the counting of flows from the same source to different destinations. [49] instead uses association mining techniques to derive the multiple flows features. In this techniques, a very important role on the attacks that could detected is played by the time window: large time window leads to ignore short pattern variations, instead small time window could lead to not having enough context to longer attacks with higher duration.

Protocol anomaly detection Another source of information in a network is the implementations of the standard protocols.
This type of NIDS rely on the fact that certain attacks does not respect the RFC specifications, it is also called specification-based intrusion detection.
The specifications can be loaded in the model manually by experts or automatically by algorithms. Both the approaches have advantages and disadvantage. Loading the specifications manually leads to a more accurate model but it requires the presence of experts and they must be uploaded frequently: moreover some protocols are proprietary and it can not be modelled. Loading specifications through algorithms coud affect the accuracy and the correctness of the protocols but it is more a flexible method.
[50] model TCP/IP standars encoding RFCs into state machines and it was able to successfully detect DoS and network probes from DARPA99 dataset.

Content anomaly detection The previous presented detection methods are mostly unable to detect client target attacks such as SQL injection and cross-site scripting, since their vector for malicious code is the payload of the packets.
Analyzing the payload is usually an expensive activity in terms of resources because it requires an in-depth search into network sessions.
The detection could be conducted using the analysis of the N-grams where an N-gram is a subsequence of N elements in a sequence. For instance, PAYL uses a vector of 1-grams that represents the frequency of each byte in the payload and through machine learning algorithms, it detects the deviation of these vectors compared to an average frequency vector. Testing it against all attacks in the DARPA 1999, get a detection rate close to 60% with a false positive rate less than 1%. [51].
Another approach is to build a model specific for critical applications: [53] builds a detector for web applications that analyze only the HTTP request URIs: it split the HTTP URI in order to obtain all the information from them and then automatically build normal models for the parameters extracted.
A more general approach is to take advantage of data preprocessing techniques to find patterns in payload without overheading: [54], after data preprocessing phase, use Snort capabilities to analyze packet payloads and find custom patterns of interest in the traffic. The alert generated was used as features to feed a Bayesian network and recognize abnormal traffic [55].

3.2 Algorithms

Anomaly detection methods is based on creating a baseline profile of the system and confronting the new input data with the profile in order to find anomalies.
Profiles usually consist on specific information about the systems: for HIDS they could be login information (frequency, origin, duration), program execution information (frequency, CPU utilization) or file access information (types of files accessed); for NIDS they could be network traffic information such as number of packets, source and destination IP addresses contacted, number of bytes exchanged, bandwidth usage [56].
To face the problem of creating a representive baseline profile and to discover with high accuracy the information that does not fit with the model, it has been proposed several methodologies that can be grouped in three main categories: statistical-based, knowledge-based and machine learning-based [21].

3.2.1 Statistical model

Statistical methods consist on maintaining two profiles during the detection: the observed profile and the previously stored baseline profile. Each time a new event occurs, the observed profile is compared to the stored baseline profile and if the 'anomaly score' exceeds a certain thresold, an alert is triggered.
This approach uses statistical computations such as probability, mean, variance and standard deviation to analyze the current profile and find anomalies [26].)
The earliest statistical approaches used an univariate model, it means that the system behaviour was molded with independent Gaussian random variables. Later, [57] and [58] proposed the multivariate models and experimental data shown that, from a combination or related variables, it is possibile to have a better discrimination level. EMERALD IDS is one of earlist NIDS based on statistical anomaly detection for providing real-time surveillance of TCP/IP based network. Also SPICE uses a statistical approach to detect stealthy scans in real-time,

assigning an anomaly score for packets based on a frequency-based mechanism [59] and [45].

Over the time, numerous other statistical approaches has been proposed, using the Markov process model or time series model but it is possible to recognise general advantages and disadvantage of this approach.
The major advantage is that is not required a prior knowledge about the normal activity of the system, it will be learnt from observations. Moreover it provides accurate detection of attacks over long periods of time. However, statistical-based approach is susceptible to be trained by an attacker in such a way that the information generated during the attacks is considered as normal.
Also, balancing false positive and false negative is a complex task since it is difficult to set the values of the different metrics. An important drawback is that it is not possible to model all the attacks by using stochastic methods [21].

3.2.2 Knowledge-based

Knowledge-based approach also called expert system, aims to classify data according with a set of rules, in particular, the normal profile is drawn according these rules: if the audit data respect the rules, then the behavior is normal, otherwise it is classified as abnormal.
A particular category of rule-based intrusion detection is the specification-based anomaly methods, for which the model is manually constructed by human experts or developed by using formal tool such as finite state machine (FSM).
It is possible to extract rules by using different techniques. Support vector machines (SVM) schemas is often used for detecting intrusions in ad-hoc networks and it is applied according different architectures [60], [61], [62]. Reputation or trust based schemas are instead often used to reduce the amount of false positive alarms in real time environments.
Knowledge-based approach is often used in ad-hoc environment for its robustness and flexibility but the main drawback is that the development of high-quality knowledge is often complex and time-consuming. [50].

3.2.3 Machine learning-based

Machine learning is the process of extracting knowledge from large quantites of data using set of rules, methods or complex functions [64].
Several machine learning-based schemas have been applied to AIDS in order to improve the accuracy and decreasing the human knowledge need; some of these are listed in the following.
Generally, there are two kinds of machine-learning algorithms: supervised and unsupervised.
[76] conducts a comparative analysis between the two kinds of machine-learning, concluding that the supervised methods achieved the best performance but the accuracy of supervised learner decreases with the presence of unknown attacks in the training phase.

Supervised algorithms Supervised learning-based IDS detect intrusions by using labeled training data. The supervised learning process includes two different phases: the training phase where the model is trained to recognize relationships between the input data and the labeled output data and the testing phase where the effectiveness of the model is evaluated.
Among the supervised method, the most used are:

- *Decision tree*: it is a tree-like graph where the nodes represent test attributes, the branches the possible values of the test and the leaves the final class labels. [65]. There are many

algorithms to build a decision tree such as ID3, C4.5, and CART [67], [66], [66], [68]. Decision tree model is used for its ability to generalize, useful in IDS because of existence of small variations of known attacks. Moreover decision tree model works very well with a large amount of data, so, for instance, it could be effective in high-speed networks [69]. The main drawback is that it is computational infeasible to create an optimal tree, and some suboptimal trees could be prone to overfitting. In literature there are studies about decision tree model for IDS such as [69].

- *K-nearest Neighbors (KNN)*: the idea behind this tecnhique it to label a data with the class of the k nearest neighbors.
  KNN is very easy to implement and to read, so analysis of results can be effective and fast; on KNN schemas can be used noise reduction techniques that work only for this algorithm and this can be effective in improving the accuracy.
  On the other hand, KNN do all the work in run-time and this can heavily affect its performance. KNN is also very sensitive to irrelevant or rendundant features [70].

- *Neural network*: it is one of the most broadly machine-learning algorithms used in IDS. It aims to simulate the operation of the human brain using a corresponding software-version of neurons and synapses.
  Their strength is the ability to recognize complex relationships between input data and labeled output data. Despite its wide use)[71], [72]), neural networks suffer in detecting less frequent attacks [26].

- *Support Vector Machines (SVM)*: it is based on mapping input data on an higher dimensions space in order to be able to split the space in a linear way. It is mainly used for its effectiveness on dataset with high number of dimensions but it is a model that is barely readable. SVM was used in the work of [79], where it is used to classify the KDD99 dataset with about 98% of accuracy.

Unsupervised algorithms Unsupervised learning-based IDS use unlabeled input datasets to train the model. [73]. So unsupervised algorithms could be an efficient way to overcome some limitations of datasets discussed in 3.1.1. Among the unsupervised methods, the most used are:

- *K-means*: it is the one of the most popular and simplest unsupervised algorithm. The aim of k-means is to separate the input data into k clusters, in which each input is selected in the cluster with the nearest mean.
  There are two important elements that could affect the accuracy of k-means algorithm: the number of k and the metric used to compute the distance between input points. Usually several solutions will be tested before choosing the most appropriate k [74] and new distance metrics has been proposed to improve accuracy [75].

- *Hierarchical clustering*: it is a clustering technique that aims to create a hierarchy of clusters following agglormerative or divisive approaches. [78] uses a hierachical clustering for IDS in order to overcome the problems of traditional clustering Also [77] use a particular hierarchical clustering to speed up the training of an SVM-based IDS.

3.3 Tools

As shown from the studies cited in this work, the research environment is industrious about the anomaly detection tecniques, developing AIDS platforms for research purpose and trying

to apply and discover new methodologies.
In spite of the numerous advantages of using AIDS, their drawbacks lead companies to still adopt the more stable signature-based IDS. Because of this, it is complex to discuss about commercial AIDS. Complete anomaly-based IDS are not yet ready to be use in real scenario, but many vendors, following the recent researches about this topic, starts to integrate some anomaly detection methods in their products.
Researches claim that the more usable and effectiveness approach is the hybrid one. Particularly for companies where attacks can leads to disastrous losses (in terms of money, data and projects), hybrid approach allows to balance the disadvantages of both signature and anomaly techniques.

In the following it will be presented a list of both commercial and open-source tools that adopt anomaly detection methods. This list does not claim to be complete and updated with all the AIDS or hybrid systems since, expecially for commercial tool, many IDS are integrated in more complex tools and the low-level details are not always discosed.

Hogzilla is an anomaly-based NIDS developed by Paulo Angelo, Gesiel Bernardes, Caio Sena and currently manteined by GuardianKey Cybersecurity.
The approach uses by this tool is mainly the statistical one but some machine-learning algorithms are used to complement the results.
In particular, during the training phase, the network traffic is used to create a baseline profile of system. This profile is composed by information related to the behavior of the hosts on the network, such as the number of hosts contacted or the ports and the number of ports used by the protected hosts.
During each detection phase, the incoming information are compared with the baseline profile in order to eventually trigger an alarm.
Moreover, the k-means clustering algorithm is used for network visibility in order to complement detection results. Hogzilla authors are also developing a new hybrid approach to the intrusion detection that use signature-based tool and machine-learning algorithms in order to be independent from the scarcity of labeled dataset during the training phase [92].
A more in-depth analysis of this tool and the test against benchmark datasets will be presented in the 3.3 chapters.

SPADE (Statistical Packet Anomaly Detection Engine) is one of the first attempt to use an anomaly detection approach for network traffic monitoring.
It was created by Stuart Staniford and Kames Hoagland whilst they worked for American Defence Advanced Research Project Agency (DARPA). This project was deleted from DARPA schedule in 2003 but it was released under GPL.
SPADE is a preprocessor plug-in for Snort that uses a simple statistical method to detect network traffic variations from a normal profile.
In more details, it saves probability tables that contain information regarding the number of occurences of different kind of packets over time. Each occurence has a weight that is the higher the more recent. Then the probabilities of the occurences are converted to an anomaly score.
SPADE sensors are setted up with an alerting threshold, so that an anomaly score could trigger an alert [86].

PAYL is a tool developed by Ke Wang and Salvatore Stolfo in 2004 to overcome the limitations of signature-based IDS.
The approach of PAYL was innovative because it was one of first IDS to analyze completely

the payload of the packets. In fact, payload analysis was very limited at the time and the more related works was PHAD and NIDES which analyze part of the payloads.
PAYL, during the training phase, builds a model of the expected payload delivered to a service during normal operations. The models consist in 1-grams that represent the byte frequency distribution of the payloads.
During the detection phase, the dector captures incoming payloads, tests them against the distribution and computing a distance metrics. Any tested payload that is too distant from the normal expected one trigger an alert [51].
At least until the 2008 this tool represents the most complete method for payload analysis, when [52] claimed that the approach use by Wang and Stolfo was ineffective on high-speed network and proposed a new analysis method called Content based payload partitioning (CPP) that improves PAYL still using 100% of the packet payload.

OPNids OPNids is a open-source IDS developed by CounterFlow AI and Deciso. It is built on top of Suricata IDS and use the DragonFly Machine Learning Engine (MLE) to collect statistics and run machine-learning algorithms on data flows.
OPNids is able to integrate and extend Suricata capability adding a new layer composed by machine-learning algorithms which processed Suricata output data [89].

3.4 Comparison with signature-based IDS

Previously, it have been presented the main traits of anomaly and signature-based IDS and how they could be implemented. In this section they will be discussed the comparision among them and it will be pointed out the strengths and the weakness of both approaches.

Given their implementations, the main differences among them is the kind of attacks that they can be recognize: generally signature-based IDS have, by definition, a database with the signatures of attacks; on the contrary anomaly-based IDS consider deviations from normal system behavior the evidences of the attacks. These approaches lead to two different results: SIDS can recognize with higher confidence the attacks, instead AIDS could does not detect the attacks at all or, in a more conservative and safety way, could raise an high amount of false positives. As already mentioned, this is caused by principles behind their implementation because, usually, the SIDS alerts are the results of a pattern matching mechanism with a well built database, instead AIDS rely only on the self-generated profiles that could be wrong (expecially in very dinamic systems). The more obvious solution to this problem for SIDS is to make sure to always have the database up to date and complete, though zero days attacks will be always undetectable with the normal implementations and the continuous updates could be very expensive in terms of time[26].

On the other hand, AIDS implementations are usually more flexible and adaptive and this lead to another result: AIDS are best in recognize uncommon or unseen attacks because unusual attacks, as well as the most common ones, generate an anomaly in the normal system behavior. Building normal system profile is quite complex operation, despite new studies and implementations: AIDS could not be effective in high dinamic systems and could even trains itself on attacks leading to consider normal an attacking behavior. Moreover, as indicated in 3.1.1, it difficult to find the right datasets to train the majority of AIDS [85].

For the previously exposed reasons, researcher aim to build hybrid solutions that could join the best of both approaches.
[80] tries to integrate signature-based Snort with an anomaly-based approach based on k-means clustering and Bayes networks in order to reduce the false alerts and still detects the unseen attacks.
[81] uses two statistical AIDS, a packet header anomaly detector PHAD [82] and network traffic anomaly detector (NETAD) [83] as preprocessors in Snort IDS and, even if it does take into account false positives rates, the number of attacks detected increases compared to using only Snort.

4 Hogzilla IDS: Overview and architecture

The recent report of WatchGuard [12] points out that about 64% of the malware evading traditional, signature-based defenses. The continuous growth of zero-day attacks highlights the limits of signature-based IDS on protecting the networks and drives the researchws on more sophisticated solutions.
In 3.3 was listed different solutions, both proprietary and open source; in this one instead, it will be study in more detail one of the open-source solution proposed: Hogzilla IDS.
Hogzilla IDS, developed by Paulo Angelo, Gesiel Bernardes, Caio Sena represents one of the few open-source available AIDS. It is a not real-time IDS that, in the current version, uses a statistical approach to the detection and a clustering algorithms is used to complement the pure detection process.
Hogzilla works using the information derived from packets sampling that are used for building histograms. Histograms are structures that hold sampling distributions of some features and are used to identify if the incoming values are atypical or not.
Moreover, a new version of Hogzilla is still under development. This versions, whose approach is explained in 4.3, aims to be a more complex hybrid solution.

4.1 Architecture

The Hogzilla architecture represented in Fig. 2 is composed of different components used for gathering information, persistence and data computation. Some scripts work as interfaces and allow the communication between the components. In the following sections will be explained in more details all the different parts that compose Hogzilla and it will explained the features obtained from the choose of this architecture.

Components (Overview) All the architecture is able to process a big amount of data using a distributed approach since it is grounded on the Hadoop Distributed File System (HDFS). HDFS is a distributed file system provided by Hadoop framework that allows to store data among different clusters.
On top of HDFS, there are Hbase, a NoSQL database that stores the data in a columnar fashion and provides a random read/write access of data stored in filesystem.
The main part of the IDS is Hogzilla component, that basing on Spark, provide all the computation necessary for taking over the intrusions.
All the notifications are stored in the HBase database and then they are displayed on Graylog, an open source log manager.

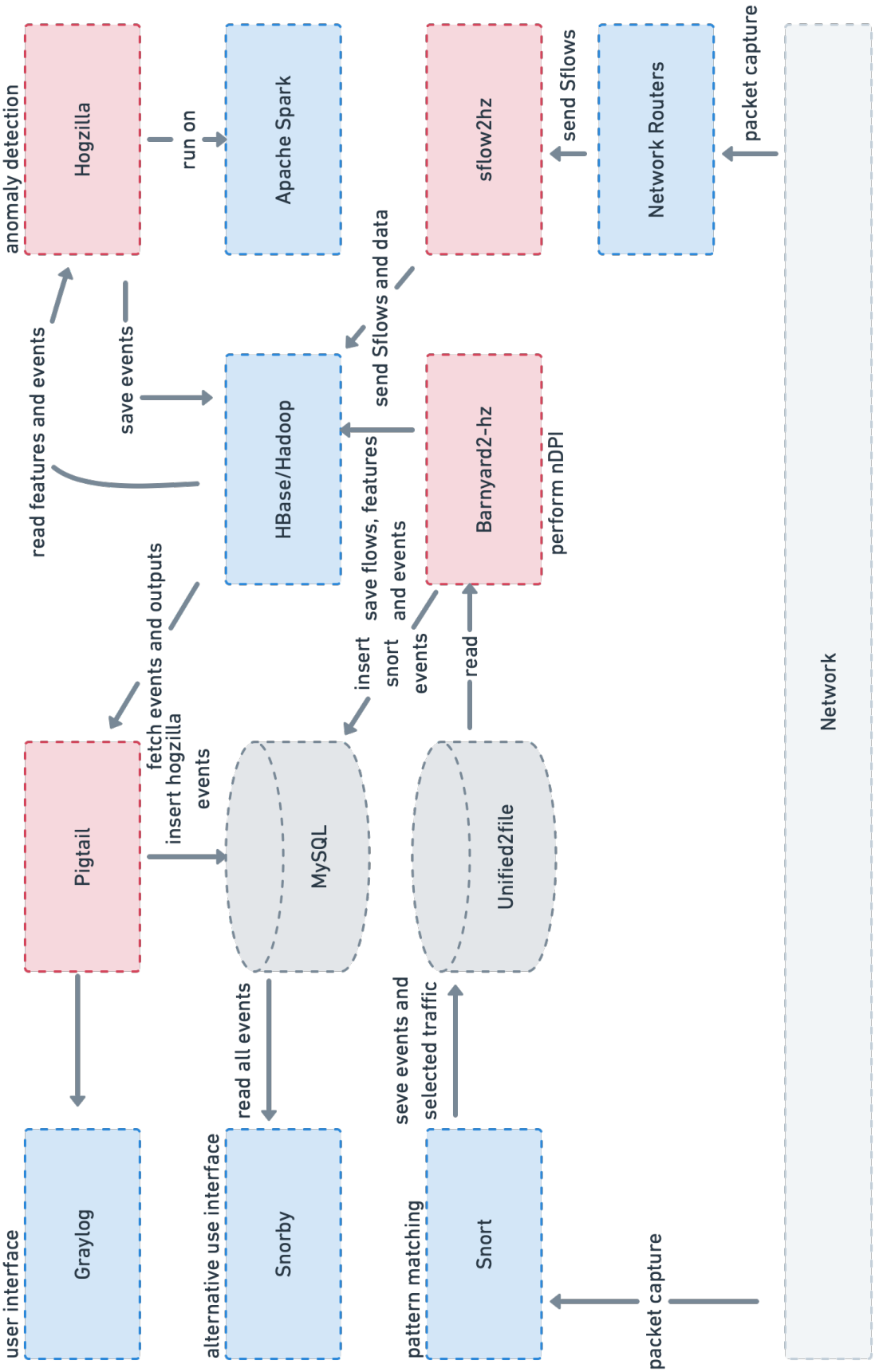


Figure 2: Hogzilla Architecture [92]

All the data needed for intrusion detection are collected using Sflow technologies which allows Hogzilla to work on high-speed network without overheading.

Features This kind of architecture shows different advantages:

- The architecture has low coupling. Even though it is necessary to rewrite or modify the scripts, it allows to change the components fairly easily since there are open-source interfaces available between the different components. This is useful to keep the architecture updated with new technologies.
- Most of the components are open source and well known, and this make the installation and the research easier.
- All the architecture is arranged in advance to work in a distributed manner and with an heavy workload. This is essential because with faster and faster connections, an IDS needs to process a lot of packets in very little time.

Below, all the components that make up Hogzilla IDS will be exposed in more detail:

4.1.1 Hadoop

Apache Hadoop [93] is a framework that can be used for the distributed processing of large datasets. Hadoop is mainly composed of three parts:

1. Hadoop Distributed File System (HDFS)
2. MapReduce pattern
3. YARN (Yet Another Resource Negotiator)

HDFS is a filesystem that manage the storage across a network of machines. It is based on a master-worker pattern: in a HDFS cluster there are the master called namenode and a certain number of worker, the datanodes.
A namenode is in charge of managing the file system tree and metadata; while the datanodes store the blocks of the files.
Furthermore, the namenode knows in which datanode each block of a certain file is located. Since a namenode must know where each block of each file is located, the HDFS 2.x allows to have different namenodes and each of them manage a particular namespace of the filesystem.
In this architecture, the client communicate with the namenode to retrieve metadata and with the datanode to read/write operations.

MapReduce is a programming model for data processing. This pattern is used with an heavy workload for distributed computing, because all the tasks can be performed in parallel. It works by breaking the processing into three phases: the map phase, the shuffle and sort phase and the reduce phase. The steps are not further explained because, MapReduce jobs are replaced with a more effective approach supplied by Spark. This topic will be analysed with more details later.
From the point of view of the architecture, for each MapReduce request from the client, the framework creates a MapReduce Job.
The MapReduce Job is divided into smaller tasks that can be called respectively Map tasks

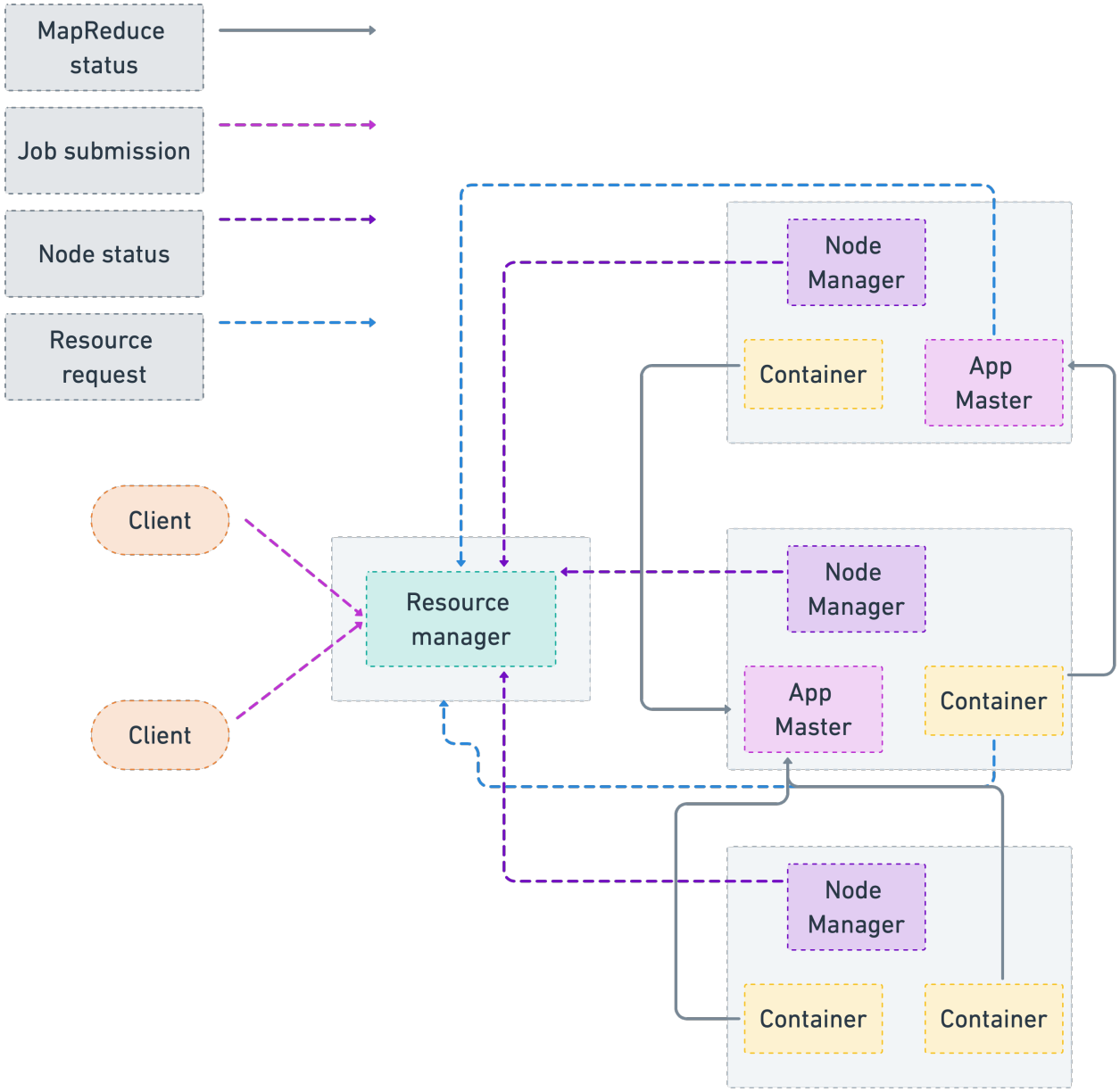


Figure 3: Hadoop architecture [100]

(splitting and mapping) and Reduce tasks (shuffling, sorting and reducing). Each tasks is then assigned to a node.
Up to the version 2.0, each job was handled by two components:

- One *JobTracker*, that run on the namenode and it is in charge of handle the lifecycle of the job and distributing the workload around the node using the data locality principle.
- One or more *Task Trackers* that run on the datanode that perform the tasks under the control of the JobTracker.

With the version 2.0, Hadoop introduces YARN (Yet Another Resource Manager).
The introduction of YARN expanded the application field of Hadoop: while with version 1.0 Hadoop was limited to run only MapReduce job, and so it was suitable only for batch processing, with version 2.0 the processing engine and the resource managing have been decoupled. This way YARN is used as the intermediate component between the HDFS and the processing

engine, allowing Hadoop to run different kind of jobs using different framework on it.

The resulting architecture is showed in the figure 3 and it is composed by the following components:

- *Resource manager*: it is a global component that run on the namenode and it is the ultimate authority that decide where the resources will be allocated among all the applications. It is composed by the:
  - *The Scheduler*: that has the role to allocate resources among the applications without monitoring or tracking the status of the applications.
  - *The Application Manager*: that is in charge of accepting jobs submissions and of negotiating the node where the Application Master will start.
- *Application Master*: it is a per-application component and it negotiate resources with the resource manager and collaborate with Node Managers to track the application status among all the node where the application is running.
- *Node Manager*: it is a per-slave component and it is responsible of monitoring and reporting the resources usage of the containers running on the node to the Resource manager.
- *Container*: allocated resources to an Application Master that are needed to run an application. Notice that an application usuallly is composed by more than one container. [100]

4.1.2 Apache Spark

Apache Spark [94] is a cluster computing framework for large-scala data processing. Unlike Hadoop, Spark does not use MapReduce as main pattern but it uses its own distributed pattern to execute the work on a cluster.
Despite this, Spark is closely integrated in Hadoop. There are three ways that can be used to deploy Spark on Hadoop:

- A *standalone deployment*: where the resources are statically allocated on the machines of on Hadoop cluster and the user can read/write data from HDFS.
- *Hadoop Yarn deployment*: thanks to the introduction of YARN, Spark can take advantage of Hadoop Resource manager, thus, it became effortless run Spark on top of YARN.
- *Spark in MR*: this is an alternative to the standalone mode, which uses SIMR to launch Spark jobs inside MapReduce.

In the case of Hogzilla, the developer choose the second option, so Hadoop runs on top of YARN.

The Spark architecture is a master/slaves architecture figured in 4 and composed by:

- *Driver program*: it lives on the master node and runs the main function of the application. It creates also the Spark Context.
- Spark Context: it is the entry point of Spark functionality; it allows to access Spark cluster with the help of the cluster manager.

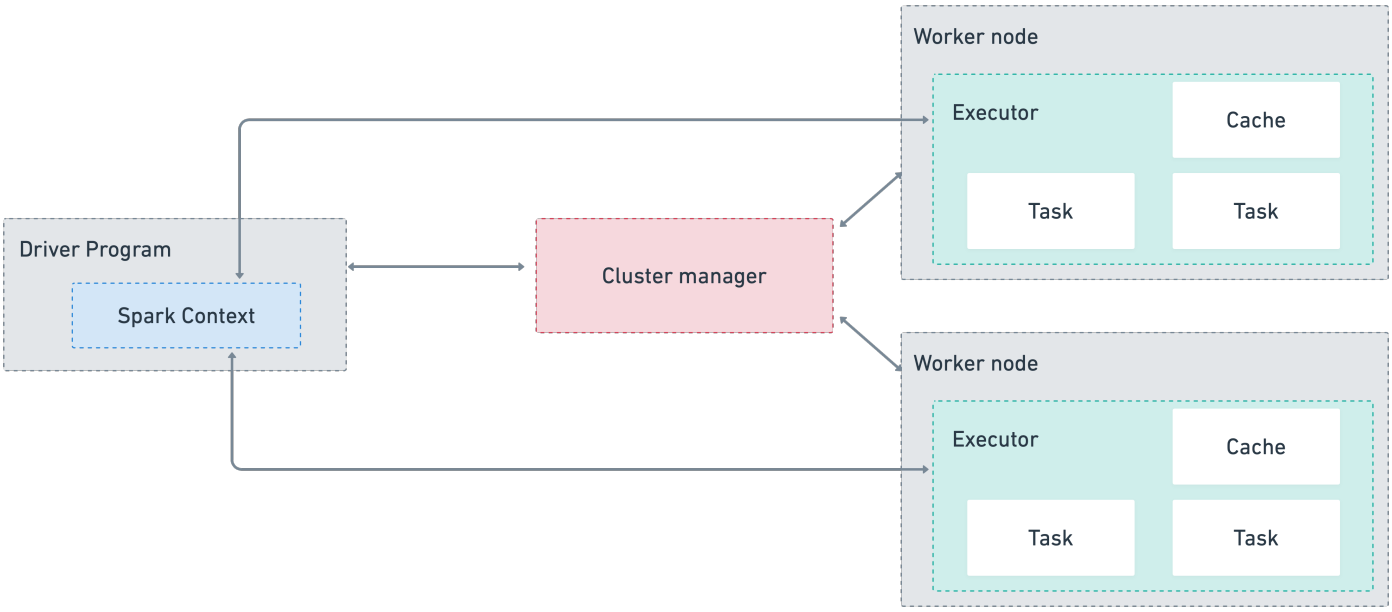


Figure 4: Spark architecture [102]

- *Cluster manager*: which allocate resources across the applications. There are many types of cluster managers that can be connected to the Spark Context depending on the architecture used: an example can be Spark own standalone cluster manager or, in case of Hadoop YARN deployment, YARN itself.
- *Executors*: they are processes launches on the worker nodes that run the tasks received from Spark Context.

The main abstraction of Spark is the Resilient Distributed Dataset (RDD). It represents a collection of element partitioned among the nodes that can be operated in parallel. The main features of this collection are:

- *Immutability*: an RDD, once created, is immutable. But even though it cannot be modified it is possible to create a new RDD by performing a transformation on an existing one.
- *Resilience*: an RDD is fault-tollerant. Since RDD tracks data origin and operations, Spark is able to recompute it.
- *Distributed*: the data present in an RDD lives on multiple nodes.
- *Lazy evaluation*: an RDD, though it has been defined, is not created until an action is called on it.

Spark provides two kind of operations on RDDs:

- *Transformation*: it generates a new RDD from an existing one.
- *Action*: perform a computation on a RDD and return the final non-RDD values (for example getting all elements in an RDD as an array or writing the result on HDFS).

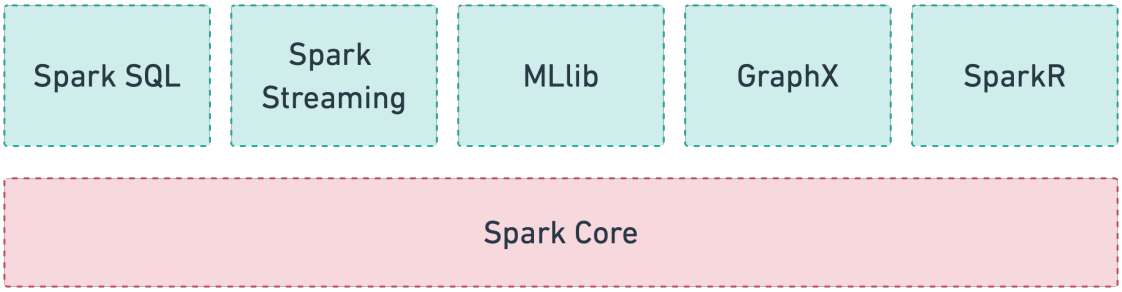


Figure 5: Spark Ecosystem [102]

In order to better understand the choice of using Spark instead of MapReduce, here it is briefly explained Spark pattern for executing the work. To describe the pattern it is necessary to introduce some specific terminology:

- *Task*: is a single operation performed on a partitions.
- *Stage*: is a sequence of tasks that can be run together without a shuffle, thus they are tasks that do not require a re-partitioning of data across the cluster. Usually this is a costly operation in terms of bandwidth usage.
- *Partition*: is logical chuck of RDD stored in a node.
- *Job*: is a sequence of stages, triggered by an action.

When an application is submitted to Spark the driver creates a directed acyclic graph (DAG) where the vertices are the RDD and the edges are the operations.
Since the lazy evaluation, only when an action is called, the DAG is converted in a physical execution plan with different stages; the units that compose the stages are the tasks. The number of tasks in a stages is at least the number of partitions of the RDD.
Driver talks with the resource manager to allocate the necessary resources for the executor on the worker node and then it send the tasks to them.

Spark has particularly been found to be faster on machine learning application than the MapReduce pattern. The reasons for this is summarised here:

- Building a DAG that describes the entire transformation of a dataset enables a more effective optimization instead of the application of numerous and consequent MapReduce functions.
- Spark is able to perform in-memory computing: the intermediate results are not stored in HDFS but they are kept in memory and only when some particular actions are used they are stored in the main memory. When the data does not fit in memory, they are recalculated or the excess data is sent to disk.

Furthermore, Spark has a rich ecosystem of components (Fig.5) that work on top of Spark Core, each of them designed for a specific data processing workload. Among these components, Hogzilla takes advantage of MLlib that provides abstractions for managing and simplifying many of the machine learning models [102].

4.1.3 HBase

Hbase [95] is a column-oriented distributed database that can run on top of HDFS. Despite HBase is not a relational database and it does not support SQL, it is able to host very large sparsed tables on clusters.
Column-oriented databases save their data grouped by columns so subsequent column values are stored contiguously on disk. The reason to adopt this model instead of a per-columns one is based on the assumption that to perform a query not all the values are needed. This assumption is particularly strong for analytical queries where aggregate operations are performed over a small number of columns.
Adopting a per-column basis offers an additional advantage on compression since the values of one columns are often very similar.
The major disadvantage with this types of database is the insertion of a new record because it has to move an high number of data.

The columns are grouped into column families and all the column belonging to same family have a common prefix. Each column has a qualifier, separated from the common prefix by a colon character (:). Physically, all column family members are stored together on the filesystem.
The cell (intersection of a row and a column) are versioned and the version is a timestamp auto-assigned by Hbase; this can be used to save multiple versions of a value.
All the rows are lexicographical sorted by their row keys that can be any array of byte. Having the row keys always sorted can be a kind of primary key index but it is also the unique because Hbase can not have more than one index in the table. This can be a problem if we want to search other field than the row key.
All the informations inside the database are automatically partitioned horizontally into regions, so a region comprises a subset of a rows.
The figure 6 represents a table in Hbase.

Regions are the units that are distributed over the HBase clusters.
Each region is assigned from the Hbase master to one region server that resides in the datanode of Hadoop architecture and each region server can handle different regions (about 1000 regions).
The region Servers are responsible for managing and executing read/write operations on a set of regions.
The Hbase master coordinates and manages the region server in the same way a namenode manages datanode in Hadoop. In particular, the Hbase Master:

- Provides an interface for creating, deleting and updating tables.
- Assigns regions to Region Servers both on startup and on recovery or load balancing operations.
- Monitors all the Region Server and performs recovery and load balancing.

In this configuration the Hbase Master is the bottleneck of the architecture: a failure on it leads to an unusable service.
For this reasons, Hbase introduces a component called ZooKeeper that acts like a coordinator inside the environment. Thanks to this, the system can have an inactive Hmaster that could replace a possible failed master.
Each component of the environment establishes a session with ZooKeeper and sends to it a heartbeat at regular intervals.
In case of failure of Hbase Master, ZooKeeper the session with it is closed and the inactive

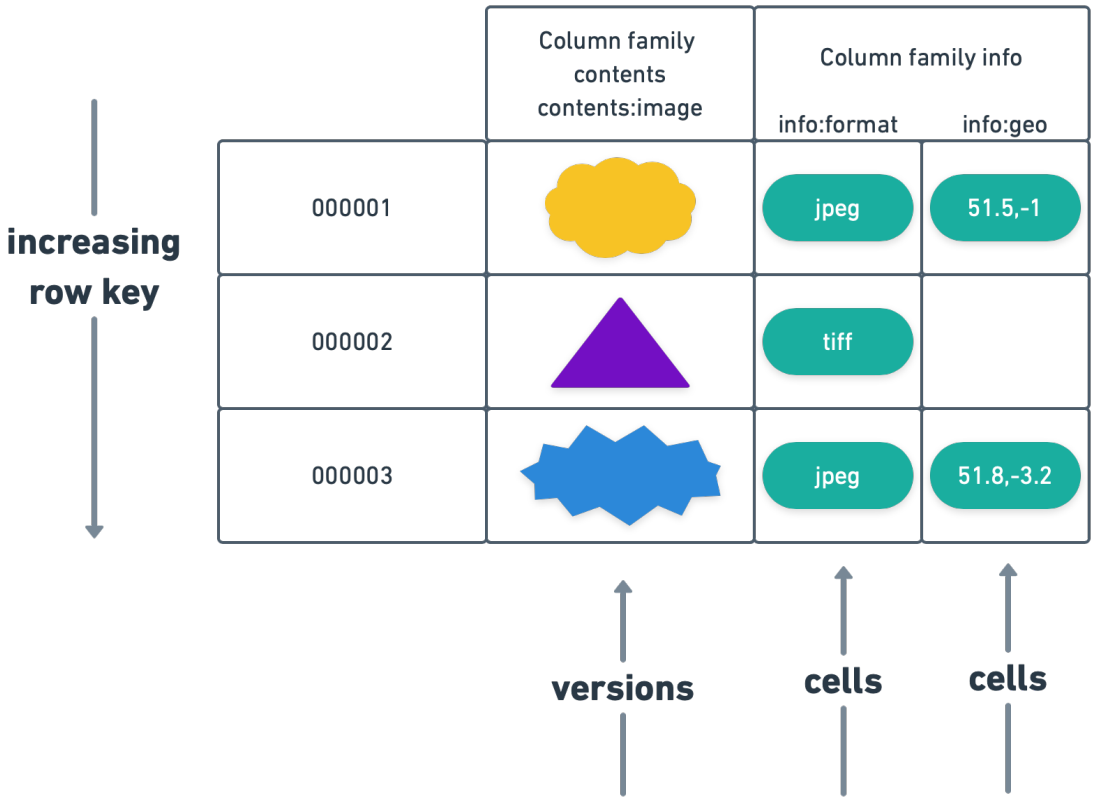


Figure 6: Hbase table [101]

Hbase Master replace it.
Also if a Region Server fails, the failure is notified by ZooKeeper to all the components and Hbase Master starts the recovery operations [101].

4.1.4 Sflow

Sflow, or Sampled Flow, [98] is a technology developed by InMon, used to monitor traffic in data networks.
It is based on sampling techniques useful in order to capture traffic statistics on network devices.
By definition, a data source is any location within a network device that can make traffic measurements, which means that it has access to a subset of the traffic flow (such as an interface).
For each data sources multiple Sflow instances can be run.
In particular, each Sflow instance is related to a packet flow instance and a counter sampling instance.

A packet flow instance is in charge of making packet sampling based on a user-defined sampling rate.
The mechanism of sampling is accomplished in this way: when a packet arrives on a network interface, the device decides whether to drop it or not; if the packet can be forwarded, a destination interface is assigned to it. At this point it is decided whether or not to sample the packet. This decision is taken based on a counter that indicates how many packets skip before sampling. When the counter reaches the zero, the packet is sampled and the counter is reset.
Another counter keeps track of the number of packets sampled within the device source. The

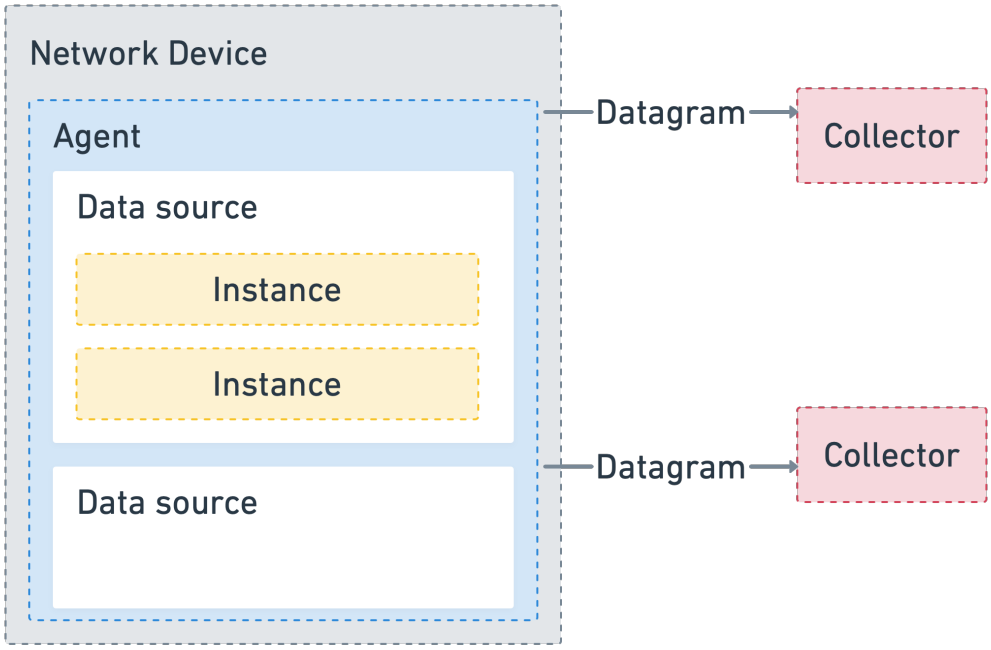


Figure 7: Sflow architecture [103]

resulted sample contains information on the packet itself such as the header, and information about the path of the packet.
The counter sampling instance is in charge of sampling, or polling, counters associated with data source. Usually a maximum sampling interval is assigned to each instance, but the Sflow agent could schedule polling to maximise internal efficiency.

The packet flow and counter sampling instances are handled on the device by the Sflow agent that provide an interface to configure the parameters and it is in charge of combining the results of both instances into an Sflow datagram.
Sflow datagram are produced by an Sflow agent and are sent to a specifiec Sflow collector using UDP protocol.
The general architecture is represented in Fig 7

The lack of reliability of UDP does not affect significantly the measurement. According to the sFlow specification [103], the developers prefer UDP over other reliable transport mechanism because of the capacity to deliver timely traffic despite possible losses.
The format of the Sflow datagram is specified using the XDR standard and the complete description of Sflow datagram can be read from the specification [103]. The information inside the datagram are layered, so they change based on the type of sample (counter or flow) and the type of packet sampled (IPv4, IPv6, ethernet frame)

Due to report statistics of sampled traffic, it is high scalable for links of speed up to 10Gb/s and beyond. In fact, collecting all the packets that flow on a links, leads to an high computational cost, specially with current speed network.
On the other hands, collecting samples cannot provide enough information about the network traffic and the IDS cannot detect properly a treat[103].

There are different technologies to perform sampling of the traffic such as NetFlow but the major advantage of Sflow rather than the competitors are:

- *The general purpose*: Sflow captures the packet header, so it is able to provide informations on all the layers, from 2 up to 7.
- *Computational costs*: Sflow does not identify the sessions, but only it only perform sampling of 1 packet over N packets. This leads to a minor computation for the devices.
- *Availability*: Sflow is integrated in network devices of all the major vendors such as Netgear, Cisco, D-Link, Fortinet.

4.1.5 Snort

Snort [97] is an open-source software that can be use as sniffer, logger or signature-based IDS depending on the configuration. In this case, we consider Snort as an IDS. Snort can be configured by placing sensors on the links of the network that we want to protect; all the information gathered pass by a complex architecture to generate eventually alerts.

Snort main components are:

- Packet Decoder
- Preprocessors
- Detection Engine
- Alerting/logging component

Packets are sniffed from the network interface during the packets acquisition. This phase is supported by Libpcap that is a cross-platform library that provides an API for receiving packets directly from the network.
Once the packets are acquired, they go through all the hierarchy of decoders that is in charge of analysing them to identify the protocol and the structure.
Preprocessors are plugin components that perform different kind of functions, with the common aim to send to the detection engine the simplest view of the traffic. Thus, for example, there is preprocessors used to reassemble fragmented packets or verify that packets are part of an established session.
Afterward, the packets go to the detection engine. Most of the capability to reveal threats, is based on the rules that compose the detection engine. In fact, the detection engine checks the data coming from the preprocessors through a a set of rules.
A rule can be syntactically analysed in two parts:

- The *header*, that identifies the type of rule (alert, log, pass, etc.), the protocol the rule is for, and the source and destination IP addresses and ports.
- The *options*, that consist in different information about the rules such as the unique identifier (SID), the message string and detection options (such as content inspection and protocol header inspection). Inside this part, it is specified what to look for in some of the fields that compose the packet structure. For example the content option allows the rule to specify a specific series of characters (or hex data) that needs to be found in the packet in order to trigger the rule.

If some packets match some rules, an alert is triggered.
The alerting component, like the preprocessor, is composed of plugins that allow the users to handle the log in different way such as saving it into a MySQL database, sending it though UNIX sockets or saving it into a log file [99].

4.1.6 Graylog

Graylog [96] is an open source log management. It is used in Hogzilla to check alerts and logs about the possibile threats and network informations.

Graylog is composed by two main components: MongoDB for storing metadata e configuration data and ElasticSearch to save data (structured or not) and quickly retrieve it using an indexing process.

4.2 Workflow

In this section it will be analysed in more details how the components previously discussed work together in the software. Portions of the scripts written by Hogzilla developers have been reported in this section to better explain how the various components communicate with each other. The description follows a bottom up approach, starting from packets acquisition, up to alert logging.

4.2.1 Monitoring

The Sflow agent on the network devices, samples the packets that arrive to the network interfaces based on user-defined sampling rates, and exports the features of the packets and of the flows the packets beloging to.
These information are sent to a collector inside UDP packets which payload contains the Sflow Datagram Structure [103].
The Sflow Datagram Structure contains information about the Sflow system for monitoring and debugging purpose (datagram source, size, Sflow version) and could contains one or more samples.

There are different types of sample such as the flow sample, and the counter sample, each with different fields.
Nonetheless, the description will be focused only on the sample dealt by Hogzilla, thus only flow sample cointaining IPv4, TCP and ICMP information.
In Tab. 2, the features extracted from a flow type sample are briefly summarised

In the collector, Hogzilla uses an open source software, called SflowTool, to interpret the Sflow datagram and forward the features in the table 2 into the *hogzilla_sflows* table using Thrift framework. For decreasing the computational costs, data are sent to Hbase in portion of 2500 samples.

4.2.2 Data storage

All the data coming from the monitor component are stored in tables of Hbase for future analysis. In the following section, the most important tables created in Hbase will be described, giving particular attention to their use and their fields.

hogzilla_sflows In the table are stored data coming from Sflow collector. The table is composed by columns belonging to the *flow* column family. Each row is identify by a primary key composed by the Sflow agent IP, the timestamp of the capture and the id of the flow between the 2500 flows. The columns beloging have the same fields of Sflow sample described 2.

Field	*Description*
sampleType	The type of sample (counter or flow)
agentID	IP address of the agent
srcMAC	MAC source address of the packet sampled
dstMAC	MAC destination address of the packet sampled
ethernetType	It is the EtherType of the ethernet frame.
inVlan	VLAN id of incoming frame
outVlan	VLAN id of outgoing frame
srcIP	IP source address of the packet sampled
dstIP	IP destination address of the packet sampled
IPprotocol	Identifies the protocol inside the IP packets
ipTos	TOS field of the IP packet sampled
ipTtl	Ttl field of the IP packet sampled
srcPort/icmp_type	TCP/UDP source port or icmp type (it depends on the protocol)
dstPort/icmp_code	TCP/UDP destination port or icmp code (it depends on the protocol)
tcpFlags	Tcp flag in hex notation
packetSize	Size of the packet
IPsize	Size of IP of the packet
samplingRate	Sampling rate of the agent

Table 2: Flow sample fields.

Field	*Description*
info:name	Name of the histogram. It is a string such as HISTAA-xx.yy.ww
Info:size	The size of the histogram
values:value1	The value of a particular feature

Table 3: hogzilla_histograms

hogzilla_histograms This table 3 contains the values of the the sampling distributions holds by histograms. Since there are different features on which made a distributions, each row is uniquely identify by a string such as *HISTAA-xx.yy.ww* where the *AA* is a number that represents the feature on which the distribution is made on and *xx.yy.ww* is the IP network 26 of the host on which is being made. The *info* columns contains information about the name of histograms, and its size. The size of an histogram is the number of values needed to make that histogram. The *values* column family contains one or more columns based on the kind of distribution. The concept of histogram is explained in more detail in 4.2.3.

hogzilla_events This table 4 contains the alerts generated after the analysis. The kind of alerts generated can be very different in terms of informations needed to identify the threat so an alert will not present all the columns listed, but only those related to its function. Furthermore, as of today, some of these columns are not used at all or their aim is not so clear but they are presented here to guarantee the integrity of the description.

4.2.3 Analysis

The data to be processed are stored in the table *hogzilla_sflows*. Data gathered by Sflow, according to the website [98], is able to detect these malicious activities:

Field	*Description*
event:sensorId	/
event:signatureId	Id of the signature (see User Manual)
event:Priorityid	/
event:text	A short text that describe the possible threat and some relevant information
event:data	Information relative to the alert. They change based on the kind of alert
event:ports	Ports involved in the possible threat
event:title	Type of alert
event:username	/
event:coords	/
event:time	Time when alert has been generated (in milliseconds)

Table 4: hogzilla_events

- *SMTP talker*: it is refererred to particular internal hosts that are sending email and should not be. It is useful to avoid email spamming too.
- *Atypical TCP port used*: this attacks is referred to alert in the case some internal hosts are using new services that have never been used before.
- *Atypical alien TCP port used*: if for two Hogzilla cycles (6h+6h), an host in external networks accesses an atypical TCP port at an internet host, it generates an alert.
- *Atypical number of pairs in the period*: an alert is sent if an host communicates with an atypical number of pairs.
- *Atypical amount of data transfered*: if a protected host sends an atypical amount of data, an alert is triggered.
- *Alien accessing too much hosts*: it refers to a general horizontal port scan.
- *P2P communication*: it generated an alert whether internal hosts use P2P communication.
- *UDP amplified (DDoS)*: if an internal host performs a DDoS attack by using amplification attacks.
- *Abused SMTP server*: it alerts if an internal host being used to send spams through e-mail servers.
- *Media streaming client*: it refers to hosts that consumes media streaming.
- *DNS tunnel*: it refers to attacks that consist on exploiting DNS traffic to tunnel malware, it is used to bypass security policies.
- *ICMP tunnel*: it refers to exploiting ICMP protocol that then can be used to bypass firewall through obfuscation.
- *Horizontal port scan*: it is an activity performed by an attacker to discover the IP addresses of the devices on a network. In this case an alert is generated if an internal hosts is used to perform an horizontal scan.
- *Vertical port scan*: it is an activity used to detect the ports open on an host and services listening on them. In this case an alert is generated if an internal hosts is used to perform an vertical scan.

- *Server under DDoS attack*: it is based on sending a large number of requests to a server from different host aiming to run out the resource of the server.
- *C&C Botnet communication*: a botnet is a network of infected devices controlled by a botmaster for malicious activities. This feature detects communication between attacker and victim hosts.

Moreover Sflow is able to perform:

- *Server grouping*: clustering servers in the protected network.
- *Network automatic inventory*: identification of the operating system.

Regarding the packet samples stored in *hogzilla_sflows*, they are aggreagated in flows, where a flow is a set of packets which represent a connection between two hosts.
Then they are transformed in an RDD, and on the RDD is performed some preparatory operations, such as applying some filter to divide the traffic in TCP/UDP and ICMP, and adding the direction of the flow (from internal to external network and viceversa).
After this preliminary process, the two resulting RDD (TCP/UDP and ICMP traffic) become the starting point on which Spark actions and transformations are applied.
The following tables describe the resulting RDD. The name used under the value column are the same of source code.

Field	*Description*
bytesUp	Bytes from inside the network to outside
bytesDown	Bytes from outside the network to inside
numberOfPkts	Number of packets
Direction	Direction of the flow.
beginTime	Timestamp of the first packet
endTime	Timestamp of the last packet
sampleRate	Sample rate of Sflow agent
Status	Status of the flow.

Table 5: TCP/UDP RDD values

In table 6 and 5 the main information about RDD for TCP/UDP traffic are presented. In particular table 6 shows the keys of the RDD whilst in table 5 a summary of the RDD values is provided.
Morevover, in table 7 a similar summary is provided for the key of RDD for ICMP traffic. About values, The only difference from 5, consists in the absence of the status field.

The detection of possible threats is based on gathering information about the connections in order to construct histograms which hold sampling distribution of some features.
So based on the historian, Hogzilla is able to detect anomalies in the traffic.
Since histograms are very important structure in this phase, a description of how they work will be provided. Histograms provide information about how the host is accessed and containes the following fields:

- *Name*: a string that identify the histogram. Since there are different features on which made a distributions, the string is the format *HISTAA-xx.yy.ww.(zz)* where the *AA* is a

Field	*Description*
myIP	IP of the host inside the network
myPort	Port of the host inside the network
alienIP	IP of the host outside the network
alienPort	Port of the host outside the network information
proto	Protocol used in the connection (TCP or UDP)

Table 6: TCP/UDP RDD key

number that represents the feature on which the distribution is made on and *xx.yy.ww.(zzz)* is the IP network of the host (only the first 24 bit) or the IP address on which is being made.

- *Size*: the size of an histogram is the number of samples that it contains.

- *Map*: it is a Scala map data structure where samples are stored. For example, if we want to build a distribution of the number of ports used by an host, the ports are the keys of the map and the number of times they have been used is the value.

Field	*Description*
myIP	IP of the host inside the network
icmpType	ICMP type
alienIP	IP of the host outside the network
icmpCode	ICMP optional code
proto	Version of ICMP.

Table 7: ICMP RDD key

For example, the histogram named *HIST02* provides information about the ports accessed by an internal host from an external host and, also in this case, for each port is associated a value that indicate the number of times that port has been used by the host, normalized on the number of flows.
During the detection, some operations may be performed on histograms.
The merge operation aims to merge two histograms: usually it is used to update the values of an histogram. This is done by merging the histogram saved on Hbase with the local histogram created during the detection process.
The merging operation consists in:

1. Joining the keys of the two histograms map

2. The size of the new histogram is the sum of both histogram size.

3. The keys that are in both histogram map have a resulting value that is the weighted average on histogram size of the two values.

4. If the size of the histogram that is save on the database is higher than 1000, a division factor of 2 is apply on it.

5. The name of the resulting histogram is the same of the histogram saved in Hbase.

There is another version of merging, called mergeMax where instead of doing the weighted average, the new value is the max of the two.

To verify if an event is typical or not, the function isTypicalEvent (Histogram.scala) is called. This function accepts two parameter: the histogram, and a string that represent the event.
If the event is key of the histogram (in the sense that exist a key belong to the map of that histogram), and if its value is greater than a certain static thresold, thus the event is typical.

Since the analysis is very similiar in most of the listed activities, a brief explanation will be carried out on how the RDDs are processed in some of those activities.

Horizontal port scan An horizontal port scan is an activity performed by an attacker to discover the IP addresses of the devices on a network.

In this case, Hogzilla attempts to reveal horizontal scan performed by internal network because it may be a policy violations or may be a malware activitiy.
The detection of this attack is based on consulting the histogram named *HIST07-xx.yy.ww.zz* where *xx.yy.ww.zz* is the IP address of the host inside the network.
This histogram contains a key for each port to which the host is connected (alienPort), and the value is the number of distinct alienIP accessed by the host on that port.

For each new flow, the histogram related to the myIP is consulted, and the new information related to the ports used are updated.
If the size of the histogram is high enough to be statistically significantly (greater than 100), the ports used in the currect flow are compared to the ports saved in the histogram.
If the number of connections for a certain port in the current flow is higher than the value saved in the histogram, an event is saved into *hogzilla_event*.
After the process, the histogram is updated using the mergeMax function.

Server under DDoS attack This attack is based on sending a large number of requests to a server from different hosts aiming to run out the resource of the server.

The dection in this case does not use histograms, but it based only on checking some conditions in the current flow.
For each flow is checked if the direction of the flow is from outside to inside, and if the alienIP and the alienPort are not in whitelist for DDoS.
Then the flows are grouped by myIP and alienIP and for each pairs (myIP, alienIP), if the number of flows is greater than a thresold, all the flows remained are sorted by timestamp and if, on average, the timestamp values are very close, an event is saved into *hogzilla_event*.

C&C Botnet Communication A botnet is a network of infected devices controlled by a botmaster for malicious activities.

The detection of the communication between a protected network and the C&C server is identify using a blacklist: whether a flow with a known botnet server is established, an event in saved in *hogzilla_event*.
The IP addresses of the server are stored in a file and that file is parsed from Hogzilla in order to fill the *hogzilla_reputation* table 27.

Vertical port scan A vertical port scan is an activity used to detect the ports open on an host and services listening on them. These alerts refer to vertical port scans performed by hosts inside the protected network.

The detection of this attack is based on consulting the histogram named *HIST08-xx.yy.ww.zz* where *xx.yy.ww.zz* is the IP address of the host inside the network.
This histogram contains a key for each number of protected ports visited during each detection process and the value is the weight.
For each detection process, the initial weight associated with the number of ports is 1.
So for the histogram merging process, how many more times that number of ports is reached, the lower the weight.
If during a detection process, the number of ports is higher than the minimum number of ports saved on the histogram and the weight of the minimum number of ports is higher than a certain static thresold, an event is saved into *hogzilla_event*.

Network automatic inventory This feature refers to automatic identification of the Operating Systems used by protected hosts. Hogzilla is able to identify the following operating system:

- Apple (IOS or MacOS)
- Android
- Windows
- Linux
- FreeBSD

The process is based on checking among the flows, if the host reached particular servers such as windowsupdate.microsoft.com or security.ubuntu.com.
The list of these servers are loaded in the *hogzilla_repository* 27 table during the initialitation of Hogzilla.
If an OS is identify using this tecniques, an event is stored in *hogzilla_inventory* table 28.

Server grouping This feature aims to identify severs in the protected network and automatically group them based on its regular use using K-means clustering.

All the information needed to cluster hosts of the network are provided by the histogram named *HIST01* that contains the ports served by hosts in the protected network, and for each port is associated a value that indicate the number of times that port has been used by the host, normalized on the number of flows.

The data preprocessing before applying K-means algorithm consist of

1. filtering the histograms of IP addresses belong to the protected network
2. collecting the ports used by all the hosts in a vector.
3. for each histogram, construct a vector where each element is the frequency of use of a specific port.

For example: if all the ports used by protected hosts are [22, 80, 443], at each histogram is associated a frequency vector like this: [0.212, 0.402, 0.0]. Each element of the last vector, indicates the frequency of use of a port:

- Port 22 it used with frequency 0.212.
- Port 80 it used with frequency 0.402.
- Port 443 it used with frequency 0.

K-means algorithm is applied on all the frequency vectors, so the ports are the dimensions of the algorithms, and the frequencies are the coordinates.
The number of resulting clusters is 10 and each cluster has an incremental id.
The results of K-means operations pass by a post processing: all the clusters which centroid have dimensions lower than 10 and number of members lower than 4 is discarded together with its members. The remaining information describe how the similar server has been grouped together, for example, the web servers may has been grouped in the same cluster. The information are saved into two tables, based on they are cluster informationsor information about cluster members.
In the table *hogzilla_clusters* 30 are saved information about the cluster such as the centroid, the number of members, the cluster id and the members.
In the table *hogzilla_cluster_members* 31 are saved information about the member cluster such as the IP address of the member, the cluster id it belongs to, the frequency vector and the distance from centroid.

4.2.4 Response

Hogzilla does not provide countermesures for the threats detected; it only provides an alert describing the kind of threats, and the flows that triggered them.
After the analysis process, when an alert is triggered, the information needed to describe it, are stored in the table *hogzilla_events*. Other information, like those related to the network automatic inventory are saved in other auxiliary tables 28.

These information are periodically fetched by a PHP script called Pigtail, that is in charge of retrieving data and generate messages to be sent to server that host the log manager.
Since the flexibility of this script, it is possible use different log manager. As described, Graylog is based on ElasticSearch and MongoDB for storing data and metadata regarding alerts; but Pigtail can also adds this data in a traditional MySQL database to support log manager based on this technology such as Snorby.
Graylog has an open port to receive UDP messages from the software who needs to log; so Pigtail, fetchs the information from the tables, deletes it from database and sends UDP messages to Graylog in GELF format [**?**], a log format that overcome some disadvantage of plain syslog such as payload limitations.

4.3 Future work

Hogzilla developers are working on another approaches to intrusion detection using an hybrid approach. The idea is to combine three types of IDS in order to balance their drawbacks and increase the performance. This hybrid approach uses two kind of signature-based approaches: pattern-matching and specification matching and moreover an anomaly-based approaches.

The main disadvantage of pattern matching is to recognize pattern despite the using of obfuscation techniques. It require to make frequent updated to signature database.
Instead the main drawback of specification matching methods is the fact that many normal systems does not respect the RFC directives, and so, benign traffic could considered a malicious activity.

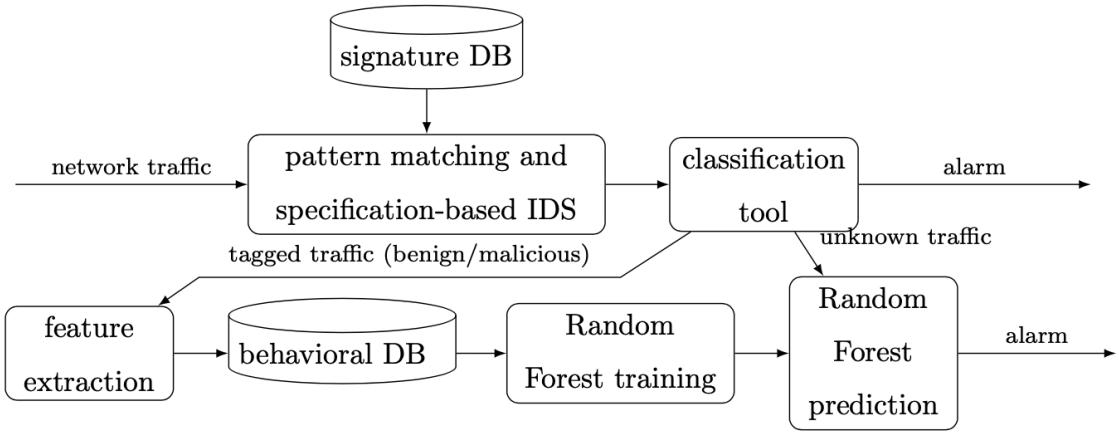


Figure 8: Approach used in under development Hogzilla project

Behavioral methods are more robust against these kind of problems but they suffers of all the problems of anomaly-based approach. One of this problem is, as previously discussed, the complex difficult to find labelled dataset to train supervised machine learning algorithms.
The solution, on which Hogzilla developers are working on, is to use specification and patter matching algorithms to classify the malicious traffic and library for application classification such as nDPI to classify the benign one.
This allow to use the labelled traffic to train the supervised algorithms.
This new approach is summarize in the figure 8.

The algorithms used is the random forest, described in 5.1.2. This approach, in particular, uses a voting scheme for the final decision of the label that can further improve the accuracy balancing the false positve.

Pratically the architecture proposed in showed in 9.

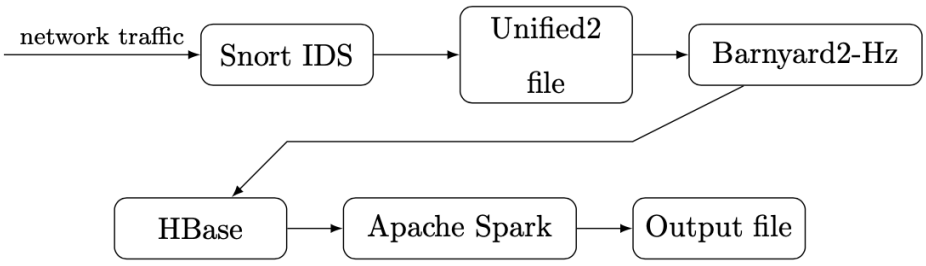


Figure 9: Architecture used in under development Hogzilla project

It include the following components:

- *Snort IDS*: is an open source signature-based IDS that detect the intrusion and label the malicious traffic.

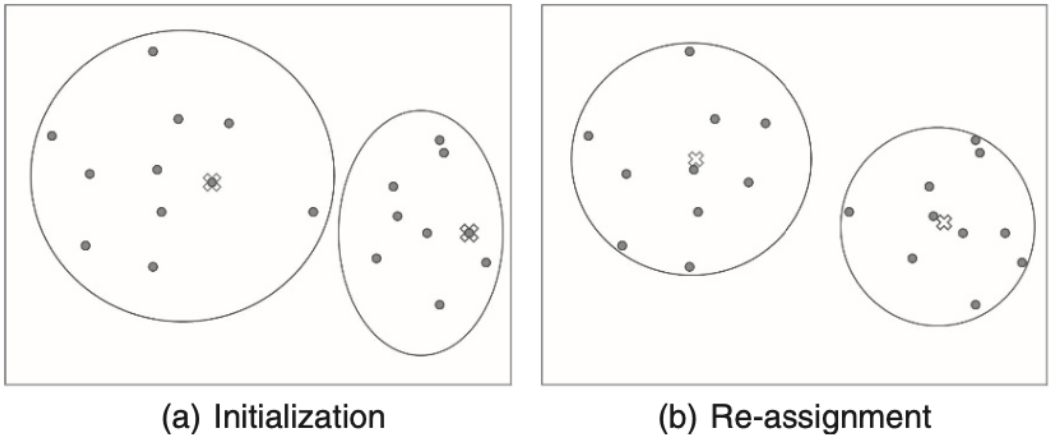


Figure 10: Example of K-means cluster computing

- *Unified2file*: is the output file of Snort detection process. It contains the events generated and the packets that triggered them.
- *Barnyard2-Hz*: it is a fork of Barnyard, a software used to read and saves the data contained in the Unified2 file into a database. In this case, the fork, through the nDPI library, extracts the flow features and saves them into the Hbase database.
- *HBase*: it is a NoSQL database used for their ability to deal with a large amount of data.
- *Apache Spark*: it is the framework on which Hogzilla runs. It is used for his performance on operations with big data.
- *Output file*: it is the file that contains the results of the routines submitted to Apache Spark.

All the above mentioned components are further discussed in 4.1.

5 Hogzilla IDS: anomaly-based traffic analysis

In this chapter, it will be explained in more details both the algorithms and the datasets used in the experimental part.

5.1 Algorithms

During the analysis phase, has been cited different machine learning algorithms used by Hogzilla to perform detections of some threats; in this section it has been explained in more detail how they works.

5.1.1 K-means

The aim of a clustering algorithm is to group homogeneous points into the same clusters. Usually the similarity is computed based on the distance among points: they are in a n-space where n is the number of features of the points and the closer they are to each other, the more

similar they are.
There are different type of clustering and different techniques, in this case it will be analyze only the k-means and the variant of k-means used by Hogzilla through MLlib of Spark.

The principle behind K-means is very simple: starting from a not optimal clustering, relocate the points into the cluster with the nearest center, update the center computing the mean of the points, and repeat the process of relocation and mean recompunting until a stopping criteria is reached.
Then, K-means algorithm can be very different based on the implementation: there are several techniques that try to overcome the limitations and the defects of this algorithm.
The two main problems of k-means are:

- The running time can be exponential in the worst case [87].
- The solution could be locally optimal but can be very far from the global optimal one.

Recent works figure out that the most pratical way to overcome these limitations is to work on the initialitations phases, in particular on choosing the best initial centroids.

An important step was made by Ostrovsky et al. and Arthur and Vassilvitskii which proposed a variant of k-means, called k-means++.
That variant is based on the idea that, to get a closer to optimal solution, the initial centroids should be sparse; so the initial centroid is choose random, and then, for each points is computed the distance between the points itself and the nearest center that has already been chosen. The next centroid is choosen randomly using a weighted probability distribution proportial to the distance of each point from the nearest center.
Results show that k-means++ variant leads to an *O(log k)* approximation of the optimum [84].
The disadvantage of k-means++ is that this approach is very difficult to be parallelize because the choosing of a new centroid depends from the previously chosen centroids.
For this reasons, MLlib of Spark, use a very similar versions of k-means++ but made so it can be parallelize, called k-means||.

5.1.2 Random Forest Classifier

A random forest is an ensemble learning method obtained by modeling a set of decision trees. A decision tree is a method for tasks of classification, based on building a graph where the nodes of the graph are the attributes that need to be analyze in order to correctly classify the object, the arcs represent the possibile values of an attribute and the the leafs are the possible target for the object. When it is trained with a labeled dateset, it produces a set of rules which will be used to make the predictions.
The main advantages of using a decision tree are:

- The tree can be easily postprocessed to derive logical rules.
- It can handle mixed discrete and continouous inputs.
- Robust to outliers.
- Scale well to large datasets.
- It can handle missing inputs.

On the other hands, the decision tree:

- It is unstable because of its high variance.
- It does not predict very accurately comparated to other models.

Starting from the same set of attributes, it is possible to construct different decision trees each with a different prediction accuracy.
The problem of building the optimal decision tree given a set of attributes is NP-complete[88], so usually greedy algorithms are used to compute a suboptimal decision trees taking locally optimal decisions, for example Hunt algorithm, ID3, C4.5, CART.

The decision tree model is prone to have an high variance, that is an high sensitivity to small variations on training set that could lead to model noise in the training set (overfitting problem); to overcome the problem of high variance, it is used the random forest classifier instead of an unique decision tree.
The random forest method is based on the bagging method; bagging is an abbreviation for Bootstrap Aggregation and the idea behind it is:

- Creating new smaller training sets, sampling the real one with replacement.
- Train a model for each new training set.
- Calculate the average prediction from each model.

In other words, the idea is that averaging the predictions of different, uncorrelated trees, can reduce the variance.
Random forest classifier improves this method by focusing on decision tree features: in fact, if the decision trees in the forest are very similar in terms of features used, the prediction may be the same for almost all the trees, so the variance may be still high.
The improvement produced by random forest is to get different decision trees by using only a random subset of features on the split-points instead of searching over all the features for the most local optimal split point. In this way the trees generated may be very uncorrelated and could produce different results.
Choosing the final prediction happens in different ways: making a numerical average of the different results or voting the most frequent label.

5.2 Datasets

In this section it will be described the datasets used for experimental part, in particular the topology of the network and the attacks leaded.
The choice of these datasets was made considering these factors:

- The number of threats that Hogzilla could detect on that dataset.
- The popularity of the dataset on the scientific community that allow to make comparision with other IDS.
- The topology of the network because Hogzilla, given its early release, could have some problems on particular topology 9.2.3.

Day	*Activity*
Friday 11/6/2010	Normal activity
Saturday 12/6/2010	Normal activity
Sunday 13/6/2010	Infiltrating the network from inside + normal activity
Monday 14/6/2010	HTTP Denial of Service + Normal activity
Tuesday 15/6/2010	Distributed Denial of Service using an IRC Botnet
Wednesday 16/6/2010	Normal activity
Tuesday 15/6/2010	Brute Force SSH + normal activity

Table 8: UNB ISXC 2012: attacks by day

5.2.1 UNB ISCX 2012

This dataset is provided by the Canadian Institute for Cybersecurity and consist of 7 days of traffic captures, both normal and malicious activities.
Each days of malicious traffic captures, consist of different kind of attacks, summed up in table 8.

The testbed, represented in the Fig. 11 consists of 21 hosts using different Windows versions chosen so that it would be possibile to exploit a set of known vulnerabilities. The 21 hosts are split in 4 distinct LAN to construct a real interconnected network.
There is another LAN with a NAT server, a main and secondary server: the main server is responsible for delivering the network website, providing email services, and acting as the internal name resolver. A secondary server is responsible for internal ASP.NET applications. The NAT server it the entry point of the network and it is connectd to the Internet with 2 IP addresses, one is multiplexed for the workstations accessing the Internet while the other is used only for the main server. Both the main and the NAT server running Ubuntu 10.04, instead the secondary server running Windows Server 2003. The sixth LAN is in charge to host all the necessary machines for capturing and monitoring the traffic. All the LAN are connected via a single layer 3 switch that provide the necessary layer 2 and 3 switching and mirroring of traffic.

Among the different attacks in the dataset, it will be tested only the vertical and horizontal scan attacks performed as preliminary operations during the day 13/06. The reason for this choice is that the internal users of the network are inaccessible from outside, since the NAT acts as a firewall, dropping any initial connections from outside the network, so all the attacks are performed via reverse shell from infected hosts inside the network. Since Hogzilla, in this case, is not able to detected attacks from internal hosts, except for vertical and horizontal port scan, it is not possibile to test all the attacks. In the following it will be described in more detail the traffic captures used in the test.

Infiltrating the network from inside In this day, the attackers exploit an Adobe Reader vulnerability to install a reverse shell on the victim host 192.168.1.105.
The victim host starts to communicate with the attacker 131.202.243.90:5555 .
In the same way, the attacker starts a communication with the victim hosts 192.168.2.112.
192.168.1.105 starts a SYN scan (from the packet 4264605) to the hosts belongs to 192.168.1.0/24 and 192.168.2.0/24 subnets.
192.168.2.112 starts a scan (ICMP, SYN, ACK) aganist the subnet 192.168.5.0/24.

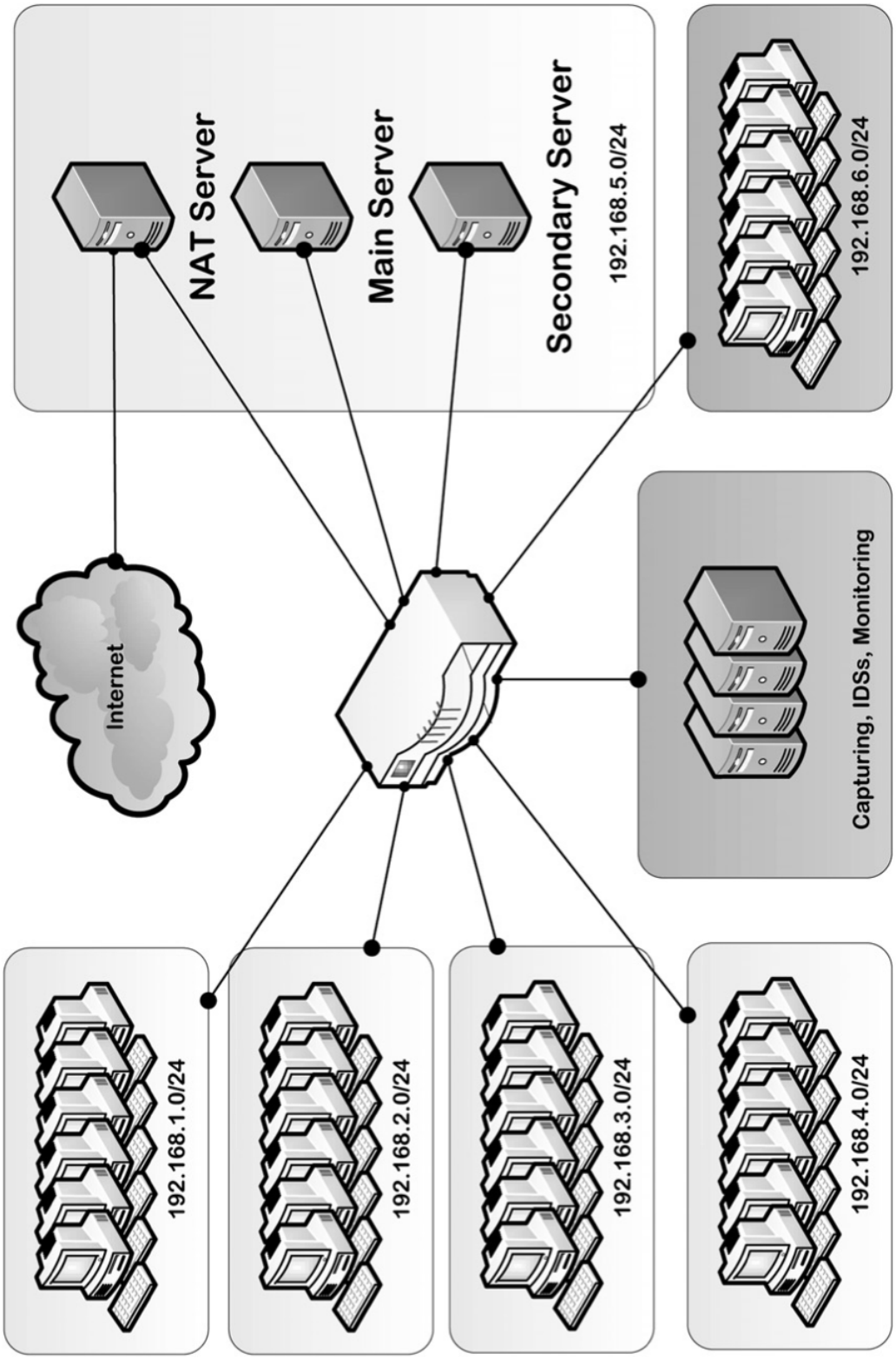


Figure 11: UNB ISCX 2012: testbed [32]

Day	*Activity*
Monday	Normal activity
Tuesday	Brute Force, FTP-Patator, SSH-Patator
Wednesday	Dos/DDos, Heartbleed
Thursday	Brute Force, XSS, Sql Injection
Friday	Botnet, Port scan, DDoS

Table 9: CICIDS 2017: attacks by day

5.2.2 CICIDS 2017

This dataset is provided by Canadian Institute for Cybersecurity and provide 5 days of captures in the network topology shown in the figure 12.
The capturing period started at 09:00 on Monday, July 3rd and continuously ran for an exact duration of 5 days, ending at 17:00 on Friday July 7th.

The testbed network is divided into 2 separate networks: the attackers network and the victim network. The Attack-Network includes one router, one switch and four PCs, which have the Kali and Windows 8.1 operating systems. The Victim-Network consists three servers, one firewall, two switches and ten PCs interconnected by a domain controller (DC) and active directory. Also, one port in the main switch of the Victim-Network has been configured as the mirror port and completely captured all send and receive traffic to the network.

There are different kind of attacks in this dataset, that are sum up in the table 9

Among these attacks, only the detectable one will be considered, thus Botnet; port scan, despite it could be detectable, is not considered because it is not performed by an internal host but by the attacker network.

The DDoS attacks of Wednesday are performed by 205.174.165.73 against the web server 205.174.165.68, but the firewall performs a NAT process so the packets are exchange between 192.168.10.50 (local IP of web server) and 172.16.0.10 (IP after NATing of the attacker). The NAT process leads to be impossible to detect DDoS attacks by using Hogzilla.

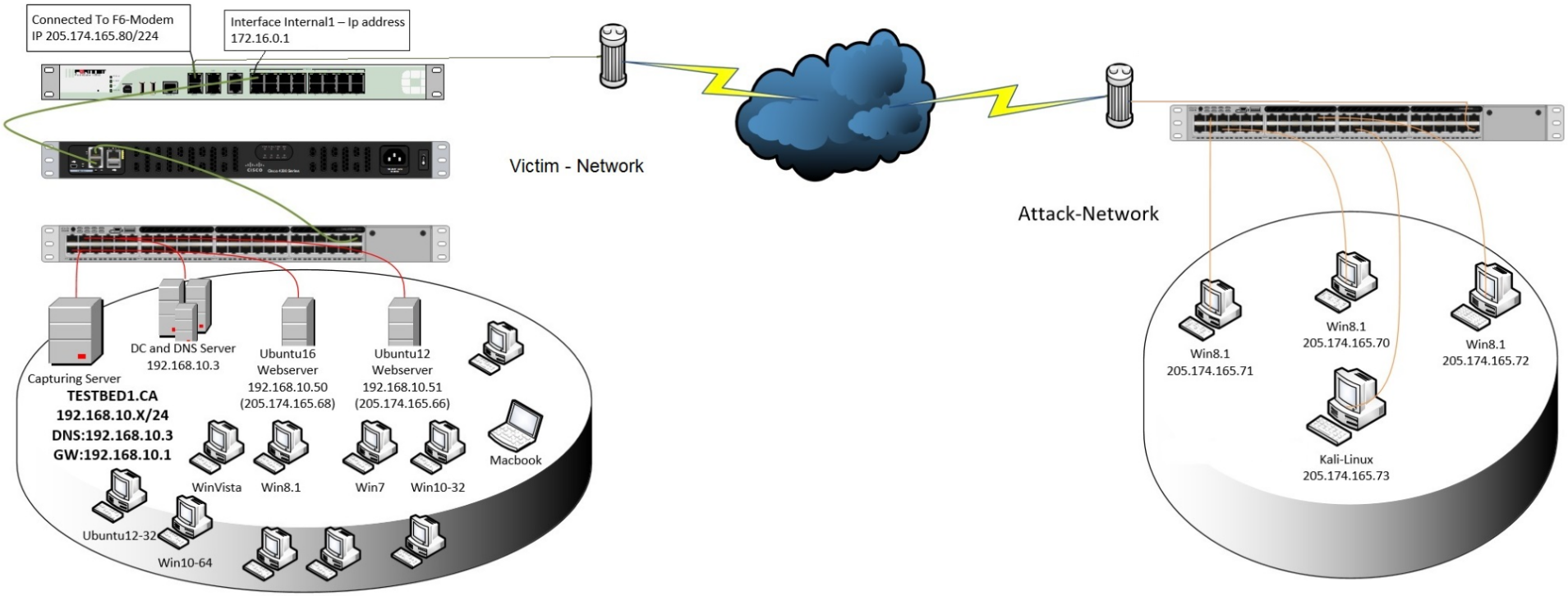
Botnet Regarding attacks using a botnet, in this dataset it is used Ares, a Python script that use HTTP for C&C. Ares is made of two main programs:

- A *command and Control server*, which is a web interface to administer the agent.
- An *agent program*, which is run on the compromised host, and ensures communication with the C&C.

With Ares, the attacker can performs different actions on the victim such as executing any commands in the victim shell, uploading file to server, downloading file through HTTP(S), taking screenshot. In this attacks, the attackers uploads images from five victim hosts.

6 Test and validation of Hogzilla features

In this section will be presented the tests conducted on Hogzilla IDS using CICIDS 2017 and UNB ISCX 2012 datasets.


Connected To F6-Modem
IP 205.174.165.80/224
Interface Internal1 – Ip address
172.16.0.1
Victim - Network
Attack-Network
Capturing Server
TESTBED1.CA
192.168.10.X/24
DNS:192.168.10.3
GW:192.168.10.1
DC and DNS Server
192.168.10.3
Ubuntu16
Webserver
192.168.10.50
(205.174.165.68)
Ubuntu12
Webserver
192.168.10.51
(205.174.165.66)
WinVista
Win8.1
Win7
Win10-32
Macbook
Ubuntu12-32
Win10-64
Win8.1
205.174.165.71
Win8.1
205.174.165.70
Win8.1
205.174.165.72
Kali-Linux
205.174.165.73

Attack	*Detection method*	*Dataset*	*Comments*
Horizontal port scan	Statistical	ISCX 2012: Sunday 13/6	Detection is not able to distinguish well the normal traffic from the attacks. This leads to an high amount of false positive especially for most used ports such as 443 and 80.
Vertical port scan	Statistical	ISCX 2012: Sunday 13/6	Detection is characterized by an high amount of false negative because the histograms on which it is based are not trained enough.
Server under DDoS attack	Rule matching	CICIDS 2017: Wednesday	Detection of this attack is affected by the erroneous direction of the packets and, since the detection is based on the number of IP addresses which contact the protected hosts, it is unusable if there is a NAT in the topology.
Server grouping	Machine learning	ISCX 2012, CICIDS 2017	It uses k-means algorithm to group the most similar servers based on the ports that they use. It is very effective but it is unable for all the services that do not use TCP protocol.
Network automatic inventory	Rule matching	ISCX 2012, CICIDS 2017	It relies on the fact that the hosts could contact some web addresses such as updatemicrosoft.com. Since these particular addresses are not reached in the testing datasets, it is not possible to test this feature.

Table 10: Attacks tested

Table 10 shows a brief summary of the attacks tested, the dataset used for testing, and the detection method used by Hogzilla. Some of these will be explained in more detail, others instead will not be discussed because they did not produce valid results.

Since both the datasets are divided by day, it will be tested only the days of datasets containing the attacks that Hogzilla can detect and listed in the 4.2.3.
In particular, it will be tested the following attacks: horizontal scan, vertical scan, and C&C botnet communication. Moreover it will test the ability of Hogzilla to do server grouping for network visibility.
All the tests are have been done splitting the single days in four parts each of which contain six hours of network traffic. Splitting is necessary since Hogzilla is not a real-time IDS but it automatically starts a new detection process every six hours. Moreover some parameters of the configuration have been changed to adapt the threshold to the sampling rate.
A phase of traning with days containing normal activities has been done before the test to allow IDS to build a baseline profile.
The detailed process of testing and the testbed are described in the 9.1.3

data class	*classified pos*	*classified neg*
pos	103	340
neg	3830	53327

Table 11: Confusion matrix of port scanning.

6.1 Port scanning

In this section will be tested Hogzilla against both vertical and horizontal scan taken by internal host against protected subnets: In particular regarding vertical scans:

- 192.168.1.105 scans some hosts belong to the subnet 192.168.2.0/24: 192.168.2.113 /.112 ./111 /.110 /.109 /.108 /.1.
- 192.168.1.105 scans some hosts belong to the subnet 192.168.1.0/24: 192.168.1.1/ .2/ .101/ .102/ .103/ .104.
- 192.168.2.112 scans some hosts belong to the subnet 192.168.5.0/24: 192.168.5.124/ .123/ .122/ .121.

Not all the above attacks has been detected, as well as not the packets belong to the detected attacks has been classified as TP. For instance, the scan against .2.112 or 1.101, was not detected.

Metrics	*Value*
Precision	2%
Recall	23%
F-score	3%
Accuracy	92%

Table 12: Metrics of port scan.

Regarding horizontal scan:

- 192.168.1.105 on the port 443 and 80 of the 192.168.2.0/24. In the PCAP file there are about 1010 packets regarding this attacks; sampling 1 packet each 100, on Hbase there are 11 packets which is what it is expected. In particular there are 7 packets with source IP address 192.168.1.105 and destination 192.168.2.0/24:80 and 4 packet with destination 192.168.2.0/24:443.
- 192.168.2.112 on the port 443 and 80 of the 192.168.5.0/24. In the PCAP file there about 12000 packets regarding this attacks: still sampling 1 packet each 100, on the Hbase there 12 packets, 8 packets with destination 192.168.5.0/24:80 and 4 packets with destination 192.168.5.0/24:443.

In the first attack, all the 7 packets with destination port 80 have been detected as attempts of horizontal scan but 2575 packets has been incorrectly classify. Instead no packets with destination 443 have been detected as attack.
In the second attack, all the 8 packets with destination port 80 have been correctly detected,

data class	*classified pos*	*classified neg*
pos	88	322
neg	13	57154

Table 13: Confusion matrix of vertical scan.

Metrics	*Value*
Precision	87%
Recall	21%
Fscore	34%
Accuracy	99%

Table 14: Metrics of vertical scan.

1242 packets have been wrongly classify as attack, instead no packets with destination port 443 have been detected.

The results can be summarize in the confusion matrix 11.

With the data resulted can be calculate the metrics in the table 12.

The results above present an evident low precision due to the high number of false positives and also a low recall due to the high number of false negatives. These two results lead to a low F-score value.
In the following, it has been conducted a separate analysis for both vertical and horizontal scan removing from the computation the packets that belong to the other attacks. From tables 15 and 13, it is possible to notice that the most false negatives derive from horizontal scan, whilst the most false positives derive from vertical scan.

Table 13 and 14 show respectively the confusion matrix and the metrics of vertical scan attack, obtained removing the horizontal scan packets from the computation.

From the results it can be notice that the precision is quite high, in fact the number of false negative is low. Instead, the recall result is not so good: given the high number of false negative, it is about of 20%. This mean that during the detection process, the majority of malicious detected packets are really malicious but most of the malicious packets are not detected.
In this case the high accuracy is not considered a valid metrics given the dataset. The number of packets belong to malicious activities is 410 over the total number of packets. Given the high unbalanced dataset, the most valid metric is the F-score that is anyway quite low because of low recall.

Table 15 and 16 show respectively the confusion matrix and the metrics of horizontal scan attack, obtained removing the vertical scan packets from the computation.

The first peculiar result is the low precision in this kind of attack given the number of false positives. The values of precision and recall claim that Hogzilla is able to classify a good number of packets belonging to this attack, but the very high number of false positive makes

data class	*classified pos*	*classified neg*
pos	15	8
neg	3817	53350

Table 15: Confusion matrix of horizontal scan.

Metrics	*Value*
Precision	0.39%
Recall	65%
Fscore	0.32%
Accuracy	93%

Table 16: Metrics of horizontal scan.

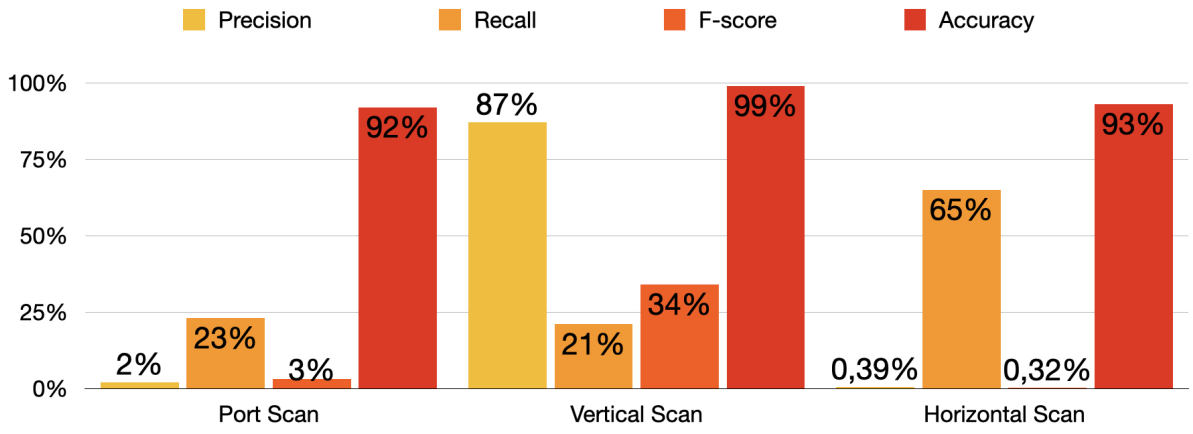


Figure 13: Comparison of metrics in port scan detection.

this information less useful because it is hidden by the misclassified packets.
Also in this case, the accuracy value can not be considered a reliable metric given the unbalanced dataset.

6.2 C&C Botnet communication

C&C Communication of the botnet present on the Friday day of CICIDS2017 regards the detection of the communication of the botnet which attacker IP address is 205.174.165.73.

At first, as in port scan analysis, it has been computed metrics considering both the attacks of the days that Hogzilla could detect and those it cannot. Table 17, shows that there are an high number of false negatives due to the attacks not detected such as DDoS and port scan from external hosts.

Due to this, the metrics presented in table 18 are not so good, with an F-score value of 3.7%. Since the only factor that affects the metrics is the number of false negatives (all due to undetectable attacks), it has been computed the confusion matrix 19 and the metrics 20 removing all the packets of undetectable threats.

In this case, the detection of communication between protected hosts and an external attackers in very accurate since all the malicious packets has been detected without any error.

data class	*classified pos*	*classified neg*
pos	177	8718
neg	0	91282

Table 17: Confusion matrix of C&C botnet communication considering all the attacks.

Metrics	*Value*
Precision	100%
Recall	1.9%
Fscore	3.7%
Accuracy	91%

Table 18: Metrics of C&C Botnet communication considering all the attacks.

As described in 4.2.3, botnet communication does not use histograms or any other statistical method, but it is detect by using a rule matching approach.

6.3 Server grouping

In this case, it will be considered the histogram named *HIST01* as data to be classify. The true positive are the histograms correctly classify as server, the true negative are the histograms correctly classify as not server, the false negatives are the server wrongly classify as normal hosts, the false positive are the host wrongly classify as servers.
Moreover, a server will be considered correctly classify only if all its services, that it is possibile to detect by Hogzilla, will be recognized in terms of ports used.

UNB ISXC 2012 As described in the 3.1.2, the testbed is composed by 3 servers:

- *NAT server*: 192.168.5.124.
- *Main server (host the network website, email service, internal name resolver)*: 192.168.5.122.
- *Secondary server, used for ASP.NET applications*: 192.168.5.123.

In this case, the main servers will be considered correctly classify only if all the services ports used will be recognized. It is an exception the internal name resolver service since the DNS query over UDP packets can not be detected by server grouping feature of Hogzilla.

The confusion matrix 21 shows that only one server is correctly classify. It is the main web server where all the services has been recognized.

- Port 22 for SSH connections.

data class	*classified pos*	*classified neg*
pos	177	0
neg	0	91282

Table 19: Confusion matrix of C&C botnet communication.

Metrics	*Value*
Precision	100%
Recall	100%
Fscore	100%
Accuracy	100%

Table 20: Metrics of C&C Botnet communication.

- Port 80 for website providing.
- Port 110 for POP protocol.
- Port 143 for IMAP protocol.

data class	*classified pos*	*classified neg*
pos	1	1
neg	0	78

Table 21: Confusion matrix of UNB ISXC 2012.

The server for ASP.NET application, instead is not correctly classify because of too few packets that lead to not have the appropriate size of the histogram.

Metrics	*Value*
Precision	100%
Recall	50%
Fscore	66%
Accuracy	98%

Table 22: Metrics of server grouping UNB ISCX 2012

According to 22 server grouping detection in this case produces good results. From the previous values it is possible to point out the discriminating ability of Hogzilla; despite the unbalanced data, Hogzilla builds two clusters, one containing only the main server, and the other containing the remaining hosts and the secondary server.
To be notice that, as previously mentioned, the undetection of secondary server is caused by the not enough packets to build a statistically relevant histograms so it is not a real error of misclassification by k-mean algorithm.
In the graph 14, it is possible to notice the two clusters: one cluster contains only the corrected server, the other cluster contains the remaining 78 hosts plus the unclassified secondary server. Notiche that the most of the hosts in the latter cluster has a zeros vector as feature vector.

CICIDS 2017 ISCX17 as has been described in 3.1.2 has three servers:

- *Domain Controller and DNS server*: 192.168.10.3.
- *Web Server*: 192.168.10.50.
- *Secondary server*: 192.168.10.51.

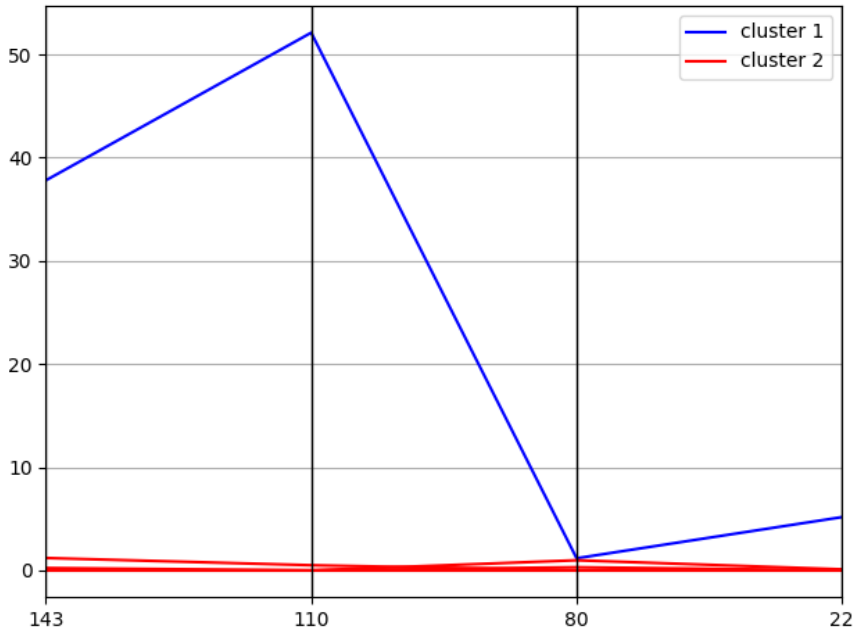


Figure 14: Parallel coordinates graph for ISCX 2012 server grouping

data class	*classified pos*	*classified neg*
pos	2	1
neg	0	36

Table 23: Confusion matrix of CICIDS 2017.

From this analysis it will be excluded the DNS service since this service can not be recognize in server grouping featuring.
It is expected instead, that domain controller service will be recognize since it uses LDAP and SMB protocol over TCP to handle the active directory.

According to 24, all the services provided by the DC server was correctly detected, together with the services offered by the web server.
The secondary web server that use the port 443 for HTTPS connections was not detected because of the low size of the corresponding histograms.

Output 15 shows that there are a cluster with the unique element the domain control server and another one, with the unique element the web server and one with all the other hosts. The DC server uses the ports 389 and 3289 for LDAP protocol, and the port 445 for SMB protocol. The web server uses the port 80 for the its service.

In this case, compared to the servers grouping of ISCX12, the value of recall is increased due to the two servers recognized over three.
Also in this case, in the case of secondary server, it is possible to notice the problem of misclassification due to the not complete histogram.

6.4 Discussion

The tests previously exposed show different performance based on the kind of attack and the approach used against it. Low accuracy was highligthed in detecting some attacks with a statistical approach such horizontal scan, whilst it had a good performance has been pointed

out in server grouping with clustering algorithm and in C&C botnet communication with rule based approach.
It should be noted also that the negative results are, in some case, attributable to the unsuitable datasets, in fact Hogzilla underlines the problems of datasets: unbalanced data and very few days of normal traffic to be used for traning, do not allow to the software to create an appropriate baseline profile for detection, affecting the results. For instance, server grouping results was affected from the lack of an appropriate training phase.
It could be useful to test it on real scenario instead of using benchmark datasets.
This problem become bigger considering the immaturity of this software. It was quite complex problem to find an appropriate benchmark dataset since the detection of some attacks are possible only in some particular cases based on the testbed topology. It was briefly mentioned the DDoS attacks case: despite it should be detected from Hogzilla, it is impossible to detect in CICIDS 2017 because attackers packets go through a NAT process that reduce the attackers IP addresses to only one. The immaturity of this software is showed also in the numerous bugs into the code as presented in 9.2.3 section, many of which required a modification into the source to work. It is needed also to upgrade the components of software, since many of them are nowdays overcome and this leading, once again a modification into the source to replace, for instance, outdated and no longer working functions.

In conclusion, Hogzilla demonstrate to have good ability in detection using a variety of approaches; some results like those obtained in botnet communication or in vertical scan are promising.
Unfortunately, it not perform very well in complex topologies or in any other envirorment different from what is was meant for, moreover it also brings with it numerous problems related to the source code. These limitations lead to the conclusion that, in this version, Hogzilla is not ready to be use in real applications.

7 Conclusions

In this work it was discussed the problem of the intrusion detection and its importance. Different technologies was proposed, revealing the complexity of the problem expecially in today interconnected world.
It was also cited numerous research that with an innovative approach mix different technologies in order to solve the intrusion detection problem.
Among these, the approach used by Hogzilla IDS was studied and tested. From the testing emerges some problems due to treat with complex topology and not balanced datasets.
These limitations indicate that Hogzilla is not ready to be tested against benchmark datasets, Better results could be reached if it is used in specific situation and feeding it with more traffic for training. Since it is necessary to modify some aspects to make it able to generalize the approach for different topologies and situations it could be considered not ready for replacing commercial tools.

Metrics	*Value*
Precision	100%
Recall	66%
Fscore	79%
Accuracy	97%

Table 24: Metrics of server grouping CICIDS 2017

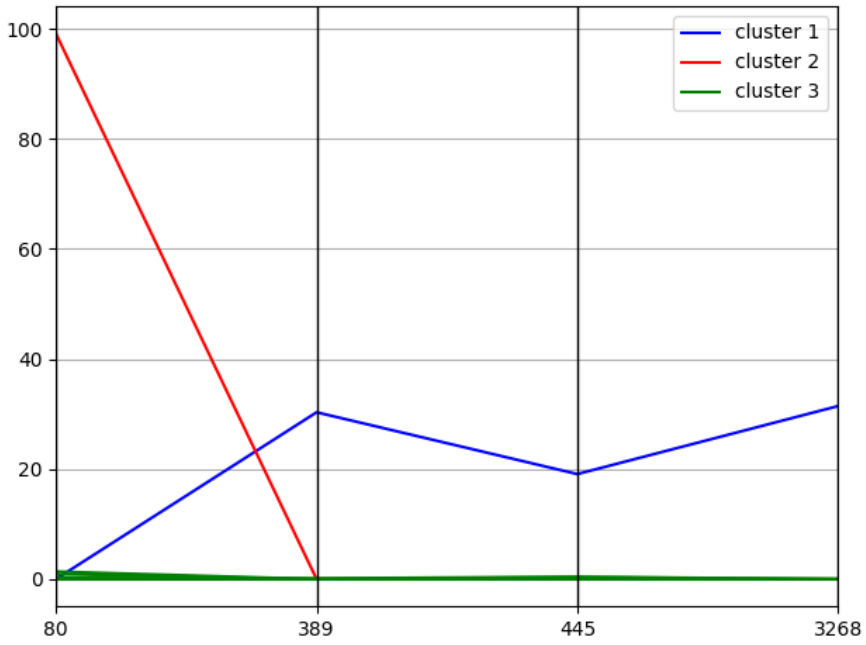


Figure 15: Parallel coordinates graph for CICIDS 2017 server grouping

Despite this, the statistical and machine-learning approaches used, abeit with some problems, are a first attempts to build an hybrid and complete IDS. Moreover the future work promises an innovative approach to intrusion detection problems.

8 Bibliography

References

[1] Cisco, "Cisco Annual Internet Report (2018-2023)", March 2020, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Australian Cyber Security Centre, "ACSC Annual Cyber Threat Report", 2019-2020, https://www.cyber.gov.au/sites/default/files/2020-09/ACSC-Annual-Cyber-Threat-Report-2019-20.pdf

[3] R. A. Kemmerer and G. Verga, "Intrusion Detection: A brief history and overview", IEEE Computer, Vol. 35, April 2002, pp. supl27 - supl30, DOI 10.1109/MC.2002.1012428

[4] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges", Elsevier Computer & Security, Vol. 28, No. 1-2, August 2008, pp. 18-28, DOI 10.1016/j.cose.2008.08.003

[5] National Institute of Standards and Technology, J. P. Anderson and P. Mell, "Computer Security Threat Monitoring and Surveillance, Technical Report", April 1980, https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/ande80.pdf

[6] D. E. Denning, "An Intrusion Detection Model", IEEE Transactions on Software Engineering, Vol. SE-13, February 1987, pp. 222-232, DOI 10.1109/TSE.1987.232894

[7] A. B. Mohamed, N. B. Idris and B. Shanmugum, "A Brief Introduction to Intrusion Detection System", Trends in Intelligent Robotics, Automation, and Manufacturing, Springer, January 2012, pp. 263-271, DOI 10.1007/978-3-642-35197-6_29

[8] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges", Springer Cybersecurity, Vol. 2, No. 1, July 2019, pp. 20, DOI 10.1186/s42400-019-0038-7

[9] A. Movaghar and F. Sabahi, "Intrusion Detection: A Survey", ICSCN-2008: 3rd Int. Conf on Systems and Networks Communications, Sliema (Malta), October 26-31, 2008, DOI 10.1109/ICSNC.2008.44

[10] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. H. Spafford and D. Zamboni, "An architecture for intrusion detection using autonomous agents", ACSAC: 14th Annual Computer Security Applications Conference, Phoenix (AZ, USA), December 7-11, 1998, DOI 10.1016/S1389-1286(00)00136-5

[11] M. K. Asif, T. A. Khan, T. A. Taj, U. Naeem and S. Yakoob, "Network Intrusion Detection and its strategic importance", BEIAC-2003: Business Engineering and Industrial Applications Colloquium, Langkawi (Malaysia), April 7-9, 2013, DOI 10.1109/BEIAC.2013.6560100

[12] WatchGuard Technologies, "Internet Security Report - Q1 2020", 2020, https://www.watchguard.com/wgrd-resource-center/security-report-q1-2020

[13] CarnagieMellow Software Engineering Institute, "State of the Practice of Intrusion Detection Technologies", March 2000, https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_16796.pdf

[14] H. Debar, M. Dacier and A. Wespi, "Towards a Taxonomy of Intrusion Detection Systems", Elsevier Computer Networks, Vol. 31, No. 8, April 1999, pp. 805-822, DOI 10.1016/S1389-1286(98)00017-6

[15] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots", ACM SIGCOMM Computer Communication Review, Vol. 34, No. 1, January 2004, pp. 51-56, DOI 10.1145/972374.972384

[16] Snort IDS, https://www.snort.org

[17] T. Lunt and R. Jagannathan, "A prototype real-time intrusion-detection expert system", IEEE Symposium on Security and Privacy, Oakland (CA, USA), April 18-21, 1988, DOI 10.1109/SECPRI.1988.8098

[18] D. S. Bauer and M. E. Koblentz, "NIDX-an expert system for real-time network intrusion detection", Computer Networking Symposium, Washington (DC, USA), 1998, DOI 10.1109/CNS.1988.4983

[19] K. Ilgun, "USTAT: a real-time intrusion detection system for UNIX", IEEE Computer Society Symposium on Research in Security and Privacy, Oakland (CA, USA), May 24-26, 1993, DOI 10.1109/RISP.1993.287646

[20] G. Vigna and R. A. Kemmerer, "NetSTAT: a network-based intrusion detection approach", 14th Annual Computer Security Applications Conference, Phoenix (AZ, USA), December 7-11, 1998, DOI 10.1109/CSAC.1998.738566

[21] V. Jyothsna and V. V. Rama Prasad, "A Review of Anomaly based Intrusion Detection Systems ", International Journal of Computer Applications, Vol. 28, No. 7, September 2011, DOI 10.5120/3399-4730

[22] T. H. Cheng, Y. Lin, Y. Lai and P. Lin, "Evasion Techniques: Sneaking through Your Intrusion Detection Prevention Systems", IEEE Communications Surveys & Tutorials, Vol. 14, No. 4, October 2011, pp. 1011-1020, DOI 10.1109/SURV.2011.092311.00082

[23] G. Varghese, J. A. Fingerhut and F. Bonomi, "Detecting evasion attacks at high speeds without reassembly", Proceedings of ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa (Italy), September 11-15, 2006, pp. 327-338, DOI 10.1145/1151659.1159951

[24] U. Payer, M. Lamberg and P. Teufl, "Hybrid Engine for Polymorphic Shellcode Detection", Second International Conference DIMVA, Vienna (Austria), July 7-8, 2005, DOI 10.1007/11506881_2

[25] G. Antichi, D. Ficara, S. Giordano, G. Procissi and F. Vitucci, "Counting Bloom Filters for Pattern Matching and Anti-Evasion at the Wire Speed", IEEE Network, Vol. 23, No. 1, January 2009, pp. 30-35, DOI 10.1007/11506881_2

[26] K. Khan, A. Mehmooda, S. Khan, M. A. Khana, Z. Iqbala and W. K. Mashwanib, "A survey on intrusion detection and prevention in wireless ad-hoc networks", Elsevier Journal of Systems Architecture, Vol. 105, May 2020, DOI 10.1016/j.sysarc.2019.101701

[27] H. Hindy, D. Brosset, E. Bayne and A. K. Seeam, "A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems", IEEE Access, Vol. 8, June 2020, DOI 10.1109/ACCESS.2017.DOI

[28] M. A. Aydin, A. H. Zaim and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security", Elsevier Computer & Electrical Engineering, Vol. 35, May 2009, pp. 517-526, DOI 10.1016/j.compeleceng.2008.12.005

[29] V. Nadiammai and M. Hemalatha, "Handling Intrusion Detection System using Snort Based Statistical Algorithm and Semi-supervised Approach", Research Journal of Applied Sciences, Engineering and Technology, Vol. 6, No. 16, September 2013, pp. 2914-2922, DOI 10.19026/rjaset.6.3672

[30] DARPA/KDD99 Dataset, https://kdd.ics.uci.edu/databases/kddcup99/task.html

[31] CAIDA website, https://www.caida.org/home/

[32] ISXCIDS 2012 dataset, https://www.unb.ca/cic/datasets/ids.html

[33] CICIDS 2017 dataset, https://www.unb.ca/cic/datasets/ids-2017.html

[34] M. Sokolova, "A systematic analysis of performance measures for classification tasks", Elsevier Information Processing & Management, Vol. 45, No. 4, June 2009, pp. 427-437, DOI 10.1016/j.ipm.2009.03.002

[35] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation", Journal of Machine Learning Technologies, Vol. 2, January 2011, pp. 2229-3981, DOI 10.9735/2229-3981

[36] W. Bulajoul, A. James and M. Pannu, "Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks", IEEE 10th International Conference on e-Business Engineering, Coventry (UK), September 11-13, 2013, pp. 168-175, DOI 10.1109/ICEBE.2013.26

[37] E. lbin and N. C. Rowe, "A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems", AINAW-2012: 26th International Conference on Advanced Information Networking and Applications Workshops, Fukuoka (Japan), March 26-29, 2012, pp. 122-127, DOI 10.1109/WAINA.2012.29

[38] S. A. Raza Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to Snort system", Elsevier Future Generation Computer Systems, Vol. 80, March 2018, pp. 157-170, DOI 10.1016/j.future.2017.10.016

[39] D. Verdejo, P. G. Teodoro and J. Tapiador, "Stochastic protocol modeling for anomaly based network intrusion detection", First IEEE International Workshop on Information Assurance, Darmstadt (Germany), March 24, 2003, pp. 3-12, DOI 10.1109/IWIAS.2003.1192454

[40] G. Mariscal, C. Fernandezand and O. Marban, "A survey of data mining and knowledge discovery process models and methodologies", The Knowledge Engineering Review, Vol. 25, June 2010, pp. 137-166, DOI 10.1017/S0269888910000032

[41] D. Jon and C. Andrew, "Data preprocessing for anomaly based network intrusion detection: A review", Elsevier Computers & Security, Vol. 30, No. 6-7, September 2011, pp. 353-375, DOI 10.1016/j.cose.2011.05.008

[42] E. Viegas, A. C. Santin and L. S. Oliveira, "Toward a Reliable Anomaly-Based Intrusion Detection in Real-World Environments", Elsevier Computer Networks, Vol. 127, August 2017, pp. 200-216, DOI 10.1016/j.comnet.2017.08.013

[43] F. Inglesias and T. Zseby, "Analysis of network traffic features for anomaly detection", Machine learning, Springer, Vol. 101, December 2014, pp. 59-84, DOI 10.1007/s10994-014-5473-9

[44] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguous and Discontiguous system call patterns", IEEE Transactions on Computers, Vol. 63, No. 4, January 2013, pp. 807-819, DOI 10.1109/TC.2013.13

[45] S. Staniford, J. Hoagland and J. McAlerney, "Practical automated detection of stealthy portscans", Journal of Computer Security, Vol. 10, No. 1-2, January 2002, pp. 105-136, DOI 10.3233/JCS-2002-101-205

[46] M. Guennoun, A. Lbekkouri and K. El-Khatib, "Selecting the Best Set of Features for Efficient Intrusion Detection in 802.11 Networks", ICTTA-2006: Information and Communication Technologies: From Theory to Applications, Damascus (Syria), April 7-11, 2006, pp. 1-4, DOI 10.1109/ICTTA.2008.4530270

[47] J. P. Early and C. Brodley, "Behavioral Features for Network Anomaly Detection", Machine Learning and Data Mining for Computer Security, Springer, 2006, pp. 107-124 DOI 10.1007/1-84628-253-5_7

[48] A. Lakhina, M. Crovella and C. Diot, "Mining anomalies using traffic feature distributions", ACM SIGCOMM Computer Communication Review, Vol. 35, No. 4, August 2005, pp. 217-228, DOI 10.1145/1090191.1080118

[49] D. Barbara, N. Wu and S. Jajodia, "Detecting Novel Network Intrusions Using Bayes Estimators", Proceedings of the 2001 SIAM International Conference on Data Mining, Chicago (IL, USA), April 2001, pp. 1-17, DOI 10.1137/1.9781611972719.28

[50] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang H and S. Zhou, "Specification-based anomaly detection: a new approach for detecting network intrusions", CCS02: Proceedings of the 9th ACM conference on Computer and communications security, New York (NY, USA), April 2002, pp. 265-74, DOI 10.1145/586110.586146

[51] K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection", RAID: International Workshop on Recent Advances in Intrusion Detection, Sophia-Antipolis (France), September 15-17, 2004, pp 203-222, DOI 110.1007/978-3-540-30143-1_11

[52] S. A. Thorat, A. K. Khandelwal, B. Bruhadeshwar and K. Kishore, "Payload Content based Network Anomaly Detection", ICADIWT: First International Conference on the Applications of Digital Information and Web Technologies, Ostrava (Czech Republic), August 4-6, 2008, pp. 127-132, DOI 10.1109/ICADIWT.2008.4664331

[53] G. Vigna and C. Kruegel, "Anomaly Detection of Web-based Attacks", Proceedings of the 10th ACM conference on Computer and communications security, New York (NY, USA), September 2003, pp. 251-61, DOI 10.1145/948109.948144

[54] W. Tylman, "Anomaly-Based Intrusion Detection Using Bayesian Networks", DepCos-RELCOMEX: Third International Conference on Dependability of Computer Systems, Szklarska Poreba (Poland), July 26-28, 2008, DOI 10.1109/DepCoS-RELCOMEX.2008.52

[55] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review", Elsevier Computers & Security, Vol. 30, No. 6-7, September-October 2011, pp. 353-375, DOI 10.1016/j.cose.2011.05.008

[56] J. Cannady and J. Harrell, "A Comparative Analysis of Current Intrusion Detection Technologies", CCE Faculty Proceedings, Presentations, Speeches and Lectures, Houston, (TX, USA), Semptember 1996, https://nsuworks.nova.edu/gscis_facpres/518

[57] SRI International, D. Denning and P. Neumann, "Requirements and model for ides: a real-time intrusion detection system", August 1985

[58] N. Ye, S. M. Emran, Q. Chen and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection", IEEE Transactions on Computers, Vol. 51, July 2002, pp. 810-820, DOI 10.1109/TC.2002.1017701

[59] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", NIST: Proceedings of the 20th national information systems security conference, Vol. 3, Baltimore (MD, USA), October 7-10, 1997, pp. 353-365

[60] V. Vladimir, "The Nature of Statistical Learning Theory', Springer, 2013

[61] H. Deng, Q. Zeng and D. P. Agrawal, "SVM-based intrusion detection system for wireless ad hoc networks", IEEE 58th Vehicular Technology Conference, Orlando (FL, USA), October 6-9, 2003, DOI 10.1109/VETECF.2003.1285404

[62] S. R. Ganapathy, Y. Palanichamy amd K. Arputharaj, "An Intelligent Intrusion Detection System for Mobile Ad-Hoc Networks Using Classification Techniques", Advances in Power Electronics and Instrumentation Engineering, January 2011, pp. 117-122, DOI 10.1007/978-3-642-20499-9_20

[63] S. Marti, T. J. Giuli, K. Lai and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks", MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, August 2000, pp. 255-265, DOI 10.1145/345910.345955

[64] S. Dua and X. Du, "Data mining and machine learning in cybersecurity", CRC press, 2016

[65] L. Rutkowski, M. Jaworski, L. Pietruczuk and P. Duda, "Decision trees for mining data streams based on the Gaussian approximation", IEEE Transactions on Knowledge and Data Engineering, Vol. 26, No. 1, February 2013, pp. 108-119, DOI 10.1109/TKDE.2013.34

[66] J. R. Quinlan, "Induction of decision trees", Springer Machine Learning, Vol. 1, No. 1, 1986, pp. 81-106, DOI 10.1007/BF00116251

[67] J. R. Quinlan, "CR 4. 5: Programs for machine learning", Elsevier, 1992

[68] L. Breiman, " Bagging predictors. Machine Learning", Springer Machine learning, Vol. 24, No. 2, 1996, pp. 123-140, DOI 10.1023/A:1018054314350

[69] M. Kumar and M. Hanumanthappa, "Intrusion Detection System Using Decision Tree Algorithm", IEEE 14th International Conference on Communication Technology, Chengdu (China), November 9-11, 2012, pp. 629-634, DOI 10.1109/ICCT.2012.6511281

[70] P. Cunningham and S. J. Delany, "k-nearest Neighbour Classifiers: 2nd Edition", April 2020, arXiv:2004.04523

[71] J. M. Bonifacio Jr, E. S. Moreira and A. M. Cansian, "An Adaptive Intrusion Detection System Using Neural Networks", Proceedings of the IFIPTC 11, 14th International conference on information security (SEC'98), Vienna (Austria) and Budapest (Hungary), August 31 - September 2, 1998

[72] S. Han, K. Kim and S. Cho, "Evolutionary Learning Program Behavior in Neural Networks for Anomaly Detection", ICONIP-2004: Neural Information Processing, 11th International Conference, Calcutta (India), November 22-25, 2004, pp. 236-241, DOI 10.1007/978-3-540-30499-9_35

[73] M. Khanam, T. Mahboob and W. Imtiaz, "A Survey on Unsupervised Machine Learning Algorithms for Automation, Classification and Maintenance", International Journal of Computer Applications, Vol. 19, June 2015, pp. 34-39, DOI 10.5120/21131-4058

[74] D. T. Pham, S. S. Dimov and C. D. Nguyen, "Selection of K in K means clustering", Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, Vol. 19, No. 1, 2005, pp. 103-119, DOI 10.1243/095440605X8298

[75] C. Annachhatre, T. Austin and M. Stamp, "Hidden Markov models for malware classification", Journal of Computer Virology and Hacking Techniques, Vol. 11, No. 2, November 2015, pp. 55-73, DOI 10.1007/s11416-014-0215-x

[76] P. Laskov, P. Dussel, C. Schafer and K. Rieck, "Learning Intrusion Detection: Supervised or Unsupervised?", ICIAP-2005: 13th International Conference, Cagliari (Italy), September 6-8, 2005, pp. 50-57, DOI 10.1007/11553595_6

[77] L. Khan, M. Awad and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering", The VLDB Journal, Vol. 16, No. 4, August 2006, pp. 507-521, DOI 10.1007/s00778-006-0002-5

[78] L. Hu, W. Ren and F. Ren, "An Adaptive Anomaly Detection based on Hierarchical Clustering", IEEE First International Conference on Information Science and Engineering, Nanjing (China), December 26-28, 2009, pp. 1626-1629, DOI 10.1109/ICISE.2009.225

[79] Y. Li, J. Xia, S. Zhang and J. Yan, "An efficient intrusion detection system based on support vector machines and gradually feature removal method", Expert Systems with Applications, Vol. 39, January 2012, pp. 424-430, DOI 10.1016/j.eswa.2011.07.032

[80] S. M. Hussein, "Performance Evaluation of Intrusion Detection System Using Anomaly and Signature Based Algorithms to Reduction False Alarm Rate and Detect Unknown Attacks", CSCI-2016: International Conference on Computational Science and Computational Intelligence, Las Vegas (NV, USA), December 15-17, 2016, DOI 10.1109/CSCI.2016.0203

[81] M. Ali Aydin, A. Hanlim Zaim and K. Ceylan, "A hybrid intrusion detection system design for computer network security", Elsevier Computers & Electrical Engineering, Vol. 35, No. 3, March 2009, pp. 517-526, DOI 10.1016/j.compeleceng.2008.12.005

[82] Florida Tech, M. V. Mahoney and P. K. Chan, 'PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic', April 2001

[83] M. V. Mahoney, "Network traffic anomaly detection based on packet bytes", Proceedings of the 2003 ACM symposium on Applied computing, March 2003, pp. 346-350, DOI 10.1145/952532.952601

[84] D. Arthur and S. Vassilvitskii, "k-means++:The Advantages of Careful Seeding", SODA-2007: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, January 2007, pp. 1027-1035

[85] A. K. Pathan, "The State of the Art in Intrusion Prevention and Detection", CRC Press, 2014

[86] Computer Security Online Ltd, S. Biles, "Detecting the Unknown with Snort and the Statistical Packet Anomaly Detection Engine (SPADE)", 2003

[87] D. Arthur and S. Vassilvitskii, "How Slow is the k-Means Method?", SCG-2006: Proceedings of the twenty-second annual symposium on Computational geometry, Sedona (AZ, USA), June 2006, pp. 144-153, DOI 10.1145/1137856.1137880

[88] L. Hyafil and R. Rivest, "Constructing optimal binary decision trees in NP-complete", Information Processing Letters, Vol. 5, May 1976, pp. 15-17, DOI 10.1016/0020-0190(76)90095-8

[89] The OPNids project, https://github.com/OPNids

[90] The Host Sflow tool, https://sflow.net/host-sflow-linux-config.php

[91] The Sflow Tool project, https://github.com/sflow/sflowtool

[92] Hogzilla IDS, https://ids-hogzilla.org

[93] Hadoop Project, https://hadoop.apache.org/

[94] Spark Project, https://spark.apache.org/

[95] Hbase Project, https://hbase.apache.org

[96] Graylog Project, https://www.graylog.org

[97] Snort website, https://snort.org

[98] Sflow website, https://sflow.net

[99] A. R. Backer and J. Esler, "Snort IDS and IPS Toolkit", Syngress Publishing, 2007

[100] T. White, "Hadoop: The Definitive Guide: 4th edition", O'Reilly Media, 2015

[101] L. George, "Hbase: The Definitive Guide", O'Reilly Media, 2011

[102] H. Luu, "Beginning Apache Spark 2", APress, 2018

[103] InMon Corp, "sFlow Version 5', July 2014 https://sflow.org/

9 Appedix

9.1 User's Manual

In this manual will be showed the installation and the compiling procedure of the Hogzilla and other components used for the test.
The manual will follow the notation used in [92], where the prompt '#' means that the command should be executed using user root and the prompt '$' means that the command should be executed using user hogzilla.

9.1.1 Installation

The installation of Hogzilla with all its components requires at least 8G of RAM and 50G of free space in the hard disk. Specifically, the space is necessary to store all the information gathered from Sflow during the monitoring. The installation is based on Debian 8.6.0 distro and require specific versions of the components to work, in particular:

- Debian 8.6.0
- Hogzilla 0.6.3
- Hadoop 2.7.3
- HBase 1.2.3
- Apache Spark 2.0.1
- Sflowtool 3.39

Set environment variables that point to import directories or indicate the version of the softwares used:

```
# HADOOPDATA="/home/hogzilla/hadoop_data"
# HADOOP_VERSION="2.6.5"
# HBASE_VERSION="1.2.6"
# SPARK_VERSION="1.6.3"
# SFLOWTOOL_VERSION="3.39"
# TMP_FILE="/tmp/.hzinstallation.temp"
# HZURL="http://ids-hogzilla.org"
# NETPREFIXES="10.1.,100.100."
# HADOOP_HOME="/home/hogzilla/hadoop"
# HBASE_HOME="/home/hogzilla/hbase"
# SPARK_HOME="/home/hogzilla/spark"
```

Install Java SDK. From the Oracle Java site.download Java SDK 1.8.0, create a folder under `/opt`, extract Java into the folder created, and setting Java JDK as the default JVM:

```
# mkdir /opt/jdk
# tar -zxf jdk-8u5-linux-x64.tar.gz -C /opt/jdk
#  update-alternatives --install /usr/bin/java java /opt/jdk/jdk1.8.0_05/bin/java 100
#  update-alternatives --install /usr/bin/javac javac /opt/jdk/jdk1.8.0_05/bin/javac 100
```

Verify the installation with the following command:

```
# java -version
```

The output should look like:

```
java version "1.8.0_05"
Java(TM) SE Runtime Environment (build 1.8.0_05-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode)
```

Install Thrift to allow the communication between Hbase and other components and other prerequisites. Then create the user *hogzilla* that will be used for starting the detection process:

```
# apt-get update
# apt-get install wget gawk sed ssh php5-thrift gcc automake autoconf make libthrift0 php5-cli
# useradd -s '/bin/bash' -m hogzilla
# su hogzilla # As user hogzilla
```

Make some configurations: create a RSA key for *hogzilla* user, create folders to host the component installation file and write variables that point to important folder to `~/.bashrc`.

```
$ mkdir /home/hogzilla/.ssh
$ ssh-keygen -t rsa -f /home/hogzilla/.ssh/id_rsa -q -N ''
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
$ ssh-keyscan localhost >> ~/.ssh/known_hosts
$ ssh-keyscan 0.0.0.0 >> ~/.ssh/known_hosts
```

```
$ mkdir /home/hogzilla/app
$ mkdir /home/hogzilla/bin
$ mkdir -p $HADOOPDATA
$ echo 'export HADOOP_HOME=/home/hogzilla/hadoop'  >> ~/.bashrc
$ echo 'export HBASE_HOME=/home/hogzilla/hbase' >> ~/.bashrc
$ echo 'export SPARK_HOME=/home/hogzilla/spark' >> ~/.bashrc
$ echo 'export HADOOP_MAPRED_HOME=$HADOOP_HOME' >>~/.bashrc
$ echo 'export HADOOP_COMMON_HOME=$HADOOP_HOME' >>~/.bashrc
$ echo 'export HADOOP_HDFS_HOME=$HADOOP_HOME' >> ~/.bashrc
$ echo 'export YARN_HOME=$HADOOP_HOME' >> ~/.bashrc
$ echo 'export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native' >> ~/.bashrc
$ echo 'export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin'  >> ~/.bashrc
$ echo 'export HADOOP_INSTALL=$HADOOP_HOME'  >> ~/.bashrc
$ echo 'export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"' >> ~/.bashrc
$ echo 'export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop' >>~/.bashrc
$ echo 'export CLASSPATH=$CLASSPATH:/home/hogzilla/hbase/lib/*' >>~/.bashrc
```

Install Hadoop, dowloding the right version (2.7.3) ans set the `/JAVA_HOME` variable:

```
$ tar xzf /home/hogzilla/app/hadoop-$HADOOP_VERSION.tar.gz -C /home/hogzilla/
$ mv /home/hogzilla/hadoop-$HADOOP_VERSION /home/hogzilla/hadoop
$ grep JAVA_HOME /etc/profile.d/jdk.sh>> $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Configure `core-site.xml`, `hdfs-site.xml` and, `yarn-site.xml` with the right address to be reached from the other components, set the directory of the namenode and of the datanode. It is possible to use the files attached into the folder `hadoop_conf`, replacing the original ones in the path `\$HADOOP_HOME/etc/hadoop/`. Then format the HDFS:

```
$ /home/hogzilla/hadoop/bin/hdfs namenode -format
```

Dowload the right version of Hbase (1.2.3),decompress it and set `/JAVA_HOME` variable:

```
$ tar xzf /home/hogzilla/app/hbase-$HBASE_VERSION-bin.tar.gz -C /home/hogzilla/
$ mv /home/hogzilla/hbase-$HBASE_VERSION /home/hogzilla/hbase
```

Configure `/hbase-site.xml`. It is possible to use the file attached into the folder `hbase_conf`, replacing the original one in the path `\$HBASE_HOME/conf/hbase-site.xml/`:

Download the right version of Apache Spark (2.0.1), decrompress it and configure `/spark-env.sh`:

```
$ tar xzf /home/hogzilla/app/spark-$SPARK_VERSION-bin-hadoop2.7.tgz -C /home/hogzilla/
$ sudo chown -R hogzilla /home/hogzilla/spark-$SPARK_VERSION-bin-hadoop2.7
```

```
3 $ mv /home/hogzilla/spark-$SPARK_VERSION-bin-hadoop2.7 /home/
    hogzilla/spark
4 $ echo 'SPARK_DRIVER_MEMORY=1G' >> $SPARK_HOME/conf/spark-env
    .sh
```

Start Hadoop and Hbase:

```
1 $ $HADOOP_HOME/sbin/start-dfs.sh
2 $ $HADOOP_HOME/sbin/start-yarn.sh
3 $ $HBASE_HOME/bin/start-hbase.sh
4 $ $HBASE_HOME/bin/hbase-daemon.sh start thrift
```

Set the rigth prefix of the network that it is necessary to protect in the `NETPREFIX` variable and execute the `hogzilla_table_generation_script.sh` attached script to generate a script for Hbase shell. This script will create the tables on Hbase.

```
1 $ $HBASE_HOME/bin/hbase shell /tmp/.hogzilla_hbase_script
```

Download Hogzilla 0.6.3 source code from the the GitHub account [1]. Make the changes on source code reported in 9.2.3. Compile the source code following the instructions in 9.1.2. Move the file .jar obtained into the right folder:

```
1 \begin{lstlisting}[xleftmargin=0.06\textwidth]
2 $mv hogzilla.jar /home/hogzilla/Hogzilla.jar
```

Install Pigtail and Hz-utils:

```
1 $ wget -c -O /home/hogzilla/app/pigtail-v1.1-latest.tar.gz "
    $HZURL/downloads/pigtail-v1.1-latest.tar.gz"
2 $ wget -c -O /home/hogzilla/app/hz-utils-v1.0-latest.tar.gz "
    $HZURL/downloads/hz-utils-v1.0-latest.tar.gz"
3 $ tar xzf /home/hogzilla/app/pigtail-v1.1-latest.tar.gz -C /
    home/hogzilla/
4 $ tar xzf /home/hogzilla/app/hz-utils-v1.0-latest.tar.gz -C /
    home/hogzilla/
```

Change configuration setting the right address for receiving alert messages on Graylog and the right version of Hbase:

```
1 $ mv -f /home/hogzilla/hz-utils/* /home/hogzilla/bin/
2 $ mkdir /usr/share/php/Thrift/Packages/
3 $ cp -a /home/hogzilla/pigtail/gen-php/Hbase//usr/share/php/
    Thrift/Packages/
4 $ sed -i.original /home/hogzilla/pigtail/pigtail.php -e "s#
    grayloghost#$GRAYLOGHOST#"
5 $ sed -i.original /home/hogzilla/bin/start-hogzilla.sh -e "s#
    HBASE_VERSION=.1.1.5.#HBASE_VERSION="$HBASE_VERSION'#"
```

Install SflowTool:

```
1 $ wget --no-check-certificate -c -O /home/hogzilla/app/
    sflowtool-$SFLOWTOOL_VERSION.tar.gz "https://github.com/
    sflow/sflowtool/releases/download/$SFLOWTOOL_VERSION/
    sflowtool-$SFLOWTOOL_VERSION.tar.gz"
```

[1] https://github.com/pauloangelo/hogzilla/

```
$ tar xzf /home/hogzilla/app/sflowtool-$SFLOWTOOL_VERSION.tar
   .gz -C /home/hogzilla/
$ cd /home/hogzilla/sflowtool-$SFLOWTOOL_VERSION ; ./
   configure
$ cd /home/hogzilla/sflowtool-$SFLOWTOOL_VERSION ; make
$ cd /home/hogzilla/sflowtool-$SFLOWTOOL_VERSION ; sudo make
   install
```

Change the versions of Hbase in the file `/start-hogzilla.sh` with the right one.

Now it is needed to Install Graylog. First, install MongoDB:

```
#apt-get install mongodb-server -y
```

Now, it is needed to install Elasticsearch, but first download the GPG key and add the repository:

```
# wget -qO - https://packages.elastic.co/GPG-KEY-
   elasticsearch | apt-key add - nano /etc/apt/sources.list.d
   /elasticsearch.list
```

And write:

```
# deb https://packages.elastic.co/elasticsearch/2.x/debian
   stable main
```

Save the file, update the repository, install Elasticsearch and start it:

```
#apt-get update -y
#apt-get install elasticsearch -y and start it
#systemctl start elasticsearch
#systemctl enable elasticsearch
```

Now install Graylog:

```
# wget https://packages.graylog2.org/repo/packages/graylog
   -2.2
-repository_latest.deb
#dpkg -i graylog-2.2-repository_latest.deb

#apt-get update -y
#apt-get install graylog-server -y
```

To configure Graylog it is needed to generate a secret and an hashed password:

```
#pwgen -N 1 -s 96
#echo -n youradminpassword | sha256sum
```

Now, modify the configuration file, adding the secret, the password and changing the web interface port. Open the configuration file:

```
#nano /etc/graylog/server/server.conf
```

And add the password and the secret to the respective variables. Then change the ports as indicated in the 9.2.3. Start Gralog:

```
#systemctl start graylog-server
#systemctl enable graylog-server
```

Add an input to Graylog for Pigtail

- Access "System-¿Inputs".
- Select "GELF UDP" in the combobox.
- Click on "Launch new input".
- Enter information like below.
  - Title: Hogzilla Events
  - Node: Select the node
  - Bind address: 0.0.0.0
  - Port: 12201
  - Receive buffer size: 262144
- Click on save

Create the following folder

```
mkdir /home/hogzilla/hogzilla
```

Download the configuration folder `conf` [2], and move in the previously created folder

Dowload *sflow2hz* [3] and put it into `/home/hogzilla/` folder

Start the services:

```
$ /home/hogzilla/bin/start-pigtail.sh
$ /home/hogzilla/bin/start-hogzilla.sh
$ /home/hogzilla/bin/start-sflow2hz.sh
$ /home/hogzilla/bin/start-dbupdates.sh
```

9.1.2 Compiling the source code

If it is necessary to modify the source code, for instance to solve some of the problems reported into 9.2.3, it was used the SBT tool [4] to compile the scala code. In order to do it, it is necessary to create a directory layout to match what SBT expects in the following way

- .idea (IntelliJ files)
- project (plugins and additional settings for sbt)
- src (source files)
  - main (application code)
    - java (Java source files)
    - scala (Scala source files) ¡– This is all we need for now
    - scala-2.12 (Scala 2.12 specific files)

[2] https://github.com/pauloangelo/hogzilla/tree/master/scripts/
[3] http://ids-hogzilla.org/downloads/sflow2hz/
[4] https://www.scala-sbt.org

  - test (unit tests)

- target (generated files)
- build.sbt (build definition file for sbt)

The `/build.sbt` file reports all the information needed to compile such as the Scala version and the dependencies. To compile Hogzilla, with the versions of the components described in the User Manual, use the `/build.st` file. Then, from the main directory, launch the commands:

```
sbt compile
sbt package
```

It creates a Java file in the `/target/scala` folder that could be move to the `/home/hogzilla`.

9.1.3 Testing process

In the following will be explained how the tests,which results are presented in 6, are conducted.

The tests has been conducted using Debian 8.6.0 distro on VMWare Fusion hypervisor software. The packets have been replayed from the same virtual host on a virtual interface created on purpose to not mix the normal traffic of the virtual host with the packets belonging to the dataset. The Sflow collector was installed into the virtual machine, using Host Sflow tool [90], that listen the interface created. Each days of the datasets was split into parts of six hours using the capinfos tool to retrieve information about the dataset and the tool editcap to split the datasets. For instance the following command:

```
capinfos -a -e input-file.pcap
```

Gives information about the start and the end time of the dataset. Basing on these information the dataset was split using the command:

```
editcap -A "starting-time" -B "ending-time" input-file.pcap
   outputfile.pcap
```

The splitting is necessary for the correct histograms building. It is also necessary to modify some parameters from the configuration file under `/home/hogzilla/hogzilla/conf/sflow.conf` with the attached file `sflow.conf`.

After the splitting, it was launched Sflowtool and Pigtail with the commands

```
./home/hogzilla/bin/start-sflows.sh
./home/hogzilla/bin/start-pigtail.sh
```

The virtual interface was created with the following commands:

```
sudo modprobe dummy
sudo ip link add eth10 type dummy
sudo ip link set eth10 up
```

The pcap file was replayed with tcpreplay tool with the following command:

```
tcpreplay -i interface-name -t input-file.pcap
```

During the replay, it is possible that due to overhead, some packets could be discarded. In this case, to reach the desired number of samples, it is necessary to lowering the speed of replaying.

At the first, all the tables were empty except *hogzilla_inventory* and *hogzilla_reputation* that was filled using the scripts `/home/hogzilla/hz-utils/genCnList.sh` and `/home/hogzilla/hz-utils/getReposList.sh` in order to get the OS information and the botnet addresses.

After replaying a part of the dataset, a detection process was launched with the command:

```
./home/hogzilla/bin/start-hogzilla.sh
```

Terminated the detection process, the table *hogzilla_sflows* was manually cleaned launching the following commands from the Hbase shell:

```
//start hbase shell
./home/hbase/bin/hbase shell
//cleaning the table
hbase(main):001:0> truncate 'hogzilla_sflows'
```

Sometimes it was necessary to save data of the tables in order to use them again without replaying again the pcap file. For instance saving the histograms after the training phases. For this reasons, it was taken a snapshot of the table with the following command in the Hbase shell:

```
hbase(main):001:0>snapshot 'hogzilla_histograms', 'snapshot-
   name'
```

And to replace the actual table with the snapshot, the following commands were launched:

```
hbase(main):001:0>disable 'hogzilla_histograms'
0 row(s) in 2.5320 seconds
hbase(main):003:0> clone_snapshot 'hogzilla_histograms', '
   snap100'
hbase(main):004:0> clone_snapshot 'snap100', '
   hogzilla_histograms'
0 row(s) in 0.4430 seconds
```

Other useful commands could be the command to count the row of a table:

```
hbase(main):005:0>count 'hogzilla_histograms'
533 row(s) in 0.2790 seconds
```

9.2 Developer's reference guide

9.2.1 Overview

The aim of the developer's reference guide is to provide a more technical and in-depth view of Hogzilla IDS. It will be explained the general architecture and how the components work and communicate each other. It will be also described the testing process that lead to the results indicated in 6 and the problems appeared during the test phase.

Hogzilla (version 0.6.3) is an open-source software written in Scala that runs over Apache Spark framework, using its library to handle data and apply machine-learning algorithms.
Hogzilla is deployed in a larger architecture that rely on the distributed file system Hadoop

YARN and includes the NoSQL Hbase database, the agent for traffic monitoring Sflows and the events logger Graylog.

The Hogzilla source code is organized in the following folders that represent also the main components that will be discussed in this chapter:

- *Cluster*: containing the definition of HogClusterMember, an object that represents a member of a cluster computed during the server grouping process.
- *Dns*: belongs to under development part.
- *Event*: it is in charge of formatting the events that come out of detection process and put them in the Hbase database.
- *Hbase*: it is in charge of all the operations regarding Hbase database such as retrieving and put information.
- *Histogram*: it contains the definion of the object HogHistogram and the operations related to them.
- *Http*: it belongs to under development part.
- *Initiate*: it is charge of initiate the operations of detection.
- *Prepare*: it contains functions that cleaning the Hbase database before a new cycle of detection.
- *Sflow*: It is the core of the detection process containing all the functions needed for discover intrusions based on histograms. It also contains the code for server clustering using k-means.
- *Util*: it contains some utility functions.

Notice that some folders will not dealt in this chapter because they belong to a still under development part.

It is more straightforward understanding how all the components in the source code and in the architecture work, following the data during the detection process and explaining in more extensive way some fundamental parts. So it will be resumed the same approach adopted in 4.2, but from a technical point of view. Notice that some folders will not dealt in this chapter because they belong to a still under development part.

It is more straightforward understanding how all the components in the source code and in the architecture work, following the data during the detection process and explaining in more extensive way some fundamental parts. So it will be resumed the same approach adopted in4.2, but from a technical point of view.

9.2.2 Detection process

Hogzilla is not a real-time IDS, but it works on 6 hours cycles: the packets flowing in the network will be sampled and retrieved all the time and stored in the Hbase database waiting for the next detection process. After a detection process, the database containg the packets sampled will be cleaned and so new packets will be hosted for the next process.

The packets are sampled by using an Sflow agent hosted usually on the router or on the switch of the network. Some features are extracted by the Sflow agent; the list of all the features is present in the 4.2.1.

sample them.The sampling rate have to be setted basing on the network speed. A good starting point for choosing the sample rate is [98], remembering that high values of sampling rate could lead to not detect attacks, but on the other hand, low values can lead to overhead. Since during the testing it was not available a network device that support Sflow technologies, an agent was installed directly on the host.
Using this software, the sample rate can be changed from the file in `/etc/hsflowd.conf`

Features of sampled packets are then send over UDP messages on the standard port 6343 to an Sflow collector for the analysis. The Sflow collector is the open-source Sflowtool [91] to 2 The collector receives UDP messages and, through the script /home/hogzilla/starts-sflows.sh is able to send data to Hbase database. The script is very simple and important line is the following:

```
1 | sflowtool -p 6343 -l | $BINPATH/sflow2hz -h 127.0.0.1 -p
  |    9090 &> \
2 |  /dev/null
```

A pipe passes data from the Sflow collector to an executable that using the Thrift interface of Hbase, send in real-time the featured gather to Hbase database.
For completeness of information, it could be said that Thrift is a software framework used for created cross-language bindings. In this case, Hbase interface allows to be accessed directly only with Java API: but it is possible to use an Hbase deamon to make Thrift work. The daemon is started using the command:

```
1 | ./hbase-deamon.sh start thrift
```

The data is received from Hbase in the table *hogzilla_sflows* that has the same structure of **??**. After that the data to be analyze are in the Hbase table *hogzilla_sflows*, a detection process will start each 6 hours from the command `/starts-hogzilla.sh`. The command initializate a new process and the logs can be displayed in the web interface of Hadoop reachable at the address `http://localhost:8088/` Hogzilla, working within Spark, communicate with Hbase through Spark libraries. The first actions taken by Hogzilla is to create a new Spark context, connect to Hbase databases and deletes all the rows older than six hours.

The detection process starts initializing the signature of the attacks and saving them into Hbase database if there are not already here. A signature identify the kind of alert among all the possible alerts [5] it is modeled with the object *HogSignature* in the file `/sflow/event/\HogSignature.scala` The table in charge of that is the table *hogzilla_signatures* having the schema in 25:

All the columns belong to *signature* family and the primary key is the signature id.

Then the configuration parameters are retrieved from the configuration file under `/home/hogzilla/hogzilla/conf/sflow.conf`. This parameters allow to tune some aspects of the detection process such as the minimum amount of flows needed for triggering an alerts, disable the detection of some attacks and so on. Before detection, it is also needed to retrieve the IP subnet to protect. It is set during the installation but it is possible to modify it acting directly on the database, in particular on the table *hogzilla_mynets* 26. This tables containes the prefixes of the network that it is needed to monitor. The primary key is the prefix and *net*

[5] https://ids-hogzilla.org/signature-db-list/

Column	*Description*
signature:id	Unique id of the signature
signature:class	Class of the signature
signature:name	Name of the signature
signature:priority	Priority of the signature
signature:revision	Not defined
signature:group_id	Not defined

Table 25: hogzilla_signtures structure

Column	*Description*
net:description	This column contains the value 'Desc' + prefix
net:prefix	Prefix of the network

Table 26: hogzilla_nets structure

is the only column family.
The prefix does not have a fixed length, thus it is possible to add entire addresses. Since it does not use masks to identify subnets, this mechanism is not similar to the longest prefix match used in the routing tables, but the match is verified using a simple string comparison.

Before detection, it is also possible to set the reputation table and the inventory table. The *hogzilla_reputation* table contains lists of particular IP addresses (auto-generated or user-defined) that could be dealt in different ways during the analysis. In particular, there are three type of lists:

- *blacklist*: used only for storing IP addresses of well-known botnet.
- *whitelist*: used for storing addresses which must to be excluded from analysis.
- *OSRepo*: report of network automatic inventory.

The table *hogzilla_reputation* contains the columns report in 27.

The *hogzilla_inventory* table 28 is used for storing the identified operating systems of the IP address belonging to the network. Each row is identified by a string composed by the IP address and the OS identified:

An example:

```
1 192.168.10.14-Windows      column=info:OS, timestamp
    =1589631329036,
2  value=Windows
```

Column	*Description*
rep:ip	IP address of the involved host
rep:list	The aim of the list such as CCBotNet, OSRepo, TT...
rep:list_type	Type of list based such as blacklist, whitelist...
rep:description	Not used

Table 27: hogzilla_reputation structure

Column	*Description*
info:title	Title to show in the log manager
info:ip	Ip address of the host
info:OS	OS of the host identified by ip

Table 28: hogzilla_inventory structure

Field	*Description*
event:sensorId	/
event:signatureId	Id of the signature (see User Manual)
event:Priorityid	/
event:text	A short text that describe the possible threat and some relevant information
event:data	Information relative to the alert. They change based on the kind of alert
event:ports	Ports involved in the possible threat
event:title	Type of alert
event:username	/
event:coords	/
event:time	Time when alert has been generated (in milliseconds)

Table 29: hogzilla_events structure

```
3 | 192.168.10.14-Windows      column=info:ip, timestamp
    =1589631329036,
4 | value=192.168.10.14
5 | 192.168.10.14-Windows      column=info:title, timestamp
    =1589631329036,
6 | value=Inventory information for 192.168.10.14
```

After setting the necessary parameters, Hogzilla connects to Hbase table *hogzilla_sflows* to retrieve features of packets and to transform each packet sampled into an RDD element. The single features belonging to each packet are transformed to be more suitable elements for the analysis, in particular the protcol, the direction and the status of each packet is identify both for TCP/UDP packets and for ICMP packets.
The two RDD (TCP/UDP and ICMP) go through the analysis process of each enabled attack detections. All the attack detection routines are composed by operations regardins filtering, map/reduce, features construction and consultation of histograms.
The routines detection of the attacks treated for the experimental part are explained in more detailed in 4.2.3.

If a detection routine trigger an alert, the event module is called (`/hogzilla/src/main/scala/org/hogzilla/event/HogEvent.scala`).
This module is in charge of saves the alerts and the correlated information into the table *hogzilla_events* of Hbase database.
The table containes the columns presented in 29:

Then the events are fetched from Pigtail component each an user-defined time. Pigtail is a PHP (`home/hogzilla/pigtail/pigtail.php`) script that is in charge of connecting with Hbase database using Thrift, fetching data from *hogzilla_events*, and then deleting them from the tables.
The fetched data are sent over UDP messages to the Graylog log manager over an user-defined

Field	*Description*
info:title	A string composed by the cluster id and the cluster centroid
info:size	The number of elements of the cluster
info:centroid	The centroid dimensions with their coordinates.
info:members	A string with the IPs of the members

Table 30: hogzilla_clusters structure

Field	*Description*
info:title	A string in the format "Group information for" + IP address
cluster:size	Number of elements into the cluster
cluster:centroid	The centroid dimensions with their coordinates.
cluster:idx	Identifier of the cluster
cluster:description	A string composed by the cluster id and the cluster centroid
member:ports	All the ports used by the IP address
member:frequencies	All the pors used by the IP address with their frequency
member:distance	Distance of the member from centroid
member:ip	IP address of the member
member:ports	Ports used by the host

Table 31: hogzilla_cluster_members structure

port. In the problems section it is reported a problem in using the default port since the new version of Hadoop uses the same port.
The message standard format is the Graylog Extended Log Format (GELF) [6].

After the detection, Hogzilla runs a routine contained in the file `/hogzilla/src/main/scala/org/hogzilla/event/HogEvent.scala` that groups servers.
The input data are fetched from the *hogzilla_histograms* tables, in particular the input data are the histograms named *HIST01*. The grouping process in explained in 4.2.3.
After the routine, the results are the clusters that are stored into the table *hogzilla_clusters* 30 and the members of each clusters, stored in the table *hogzilla_cluster_members* 31.
Also data from these tables are fetched by Pigtail and send to Graylog with the same procedures of detection process.

9.2.3 Problems encountered

During the testing phase and the installation, some problems emerged. Some of these can be resolved modifying slightly the source code, some other requires a more complex solution.

Input data error Sometimes, during the detection process, Sparks can stop and raise an input error while trying to convert timestamps values which are contained into *hogzilla_sflows*. The problem is given by the fact that the Sflow agent add space character at the end of the values, so it is not possible to convert timestamp directly in a long type.
The solution is to replace these lines (at 807 and 894 of `HogSflows.scala`)

```
1 val timestamp  = Bytes.toString(result.getValue(Bytes.toBytes
    ("flow"),
```

[6] https://docs.graylog.org/en/3.2/pages/gelf.html

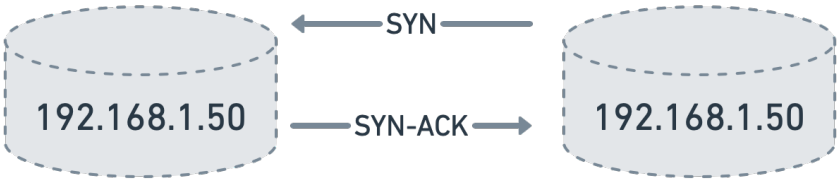


Figure 16: Example of connection

```
2 |         \Bytes.toBytes("timestamp"))).toLong
```

with:

```
1 |val timestamp  = Bytes.toString(result.getValue(Bytes.toBytes
  |   ("flow"),
2 |         \Bytes.toBytes("timestamp"))).replace("\\x00","" ).
  |            toLong
```

Uploaded functions Because of using the version 2.0.1 of Spark, it is necessary to replace the function
`toArray()`with the function `collect` into `HogSflows.scala` and `HogHbaseHistogram.scala`.

Same ports used Since by default both Graylog and Hadoop use the same port, it is necessary to change the default port of the Graylog web interface.
It is possible modifying the *http_bind_address* from 127.0.0.1:9000 to 127.0.0.1:9100 for instance.

Erroneous detected direction of the packets During the test of DDoS attack, it has been discover some specific cases where the direction of the packets was wronly assigned. The schema 16 represents the conditions apply to incoming packets to determine the direction:

The schema represents the situation where there are the host 192.168.1.50 that belong to the protected network and an external host 193.168.0.1
The external host begins a threeway handshake, sending the SYN packet, internal host replay with the SYN-ACK packet. In this case, the SYN packet is classified correctly with RIGHTLEFT direction; instead the SYN-ACK is wrongly classified as RIGHTLEFT.

It is complex to fix this problem, but not all the attacks detection routines deal with direction values so not all the routines are affected by this problem.