INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Network anomalies detection via event analysis and correlation by a smart system

Gonçalo Monteiro Cruz

Master Degree in Computer Science

Supervisor:
PhD, João Pedro Afonso Oliveira da Silva, Assistant Professor,
Iscte

Co-Supervisor:
PhD, Adriano Lopes, Invited Assistant Professor,
Iscte

October, 2022

TECNOLOGIAS
E ARQUITETURA

Department of Information Science and Technology

Network anomalies detection via event analysis and correlation by a smart system

Gonçalo Monteiro Cruz

Master Degree in Computer Science

Supervisor:
PhD, João Pedro Afonso Oliveira da Silva, Assistant Professor,
Iscte

Co-Supervisor:
PhD, Adriano Lopes, Invited Assistant Professor,
Iscte

October, 2022

To my family and friends.

Acknowledgements

I would like to thank my supervisors, Prof. João Oliveira and Prof. Adriano Lopes, for the continuous support and transmission of knowledge for the realization of this thesis.

To Eng. Dauto Jeichande and Dr. Paulo Veiga, whose knowledge contributed to the development of this thesis.

To those who participated in this research, your contributions were fundamental.

I would like to thank Iscte and ISTAR for granting me a research scholarship for the elaboration of this thesis.

To my family and friends who always gave words of support and encouragement.

Abstract

The multidisciplinary of contemporary societies compel us to look at Information Technology (IT) systems as one of the most significant grants that we can remember. However, its increase implies a mandatory security force for users, a force in the form of effective and robust tools to combat cybercrime to which users, individual or collective, are exposed almost daily. Monitoring and detection of this kind of problem must be ensured in real-time, allowing companies to intervene fruitfully, quickly and in unison.

The proposed framework is based on an organic symbiosis between credible, affordable, and effective open-source tools for data analysis, relying on Security Information and Event Management (SIEM), Big Data and Machine Learning (ML) techniques commonly applied for the development of real-time monitoring systems. Dissecting this framework, it is composed of a system based on SIEM methodology that provides monitoring of data in real-time and simultaneously saves the information, to assist forensic investigation teams. Secondly, the application of the Big Data concept is effective in manipulating and organising the flow of data. Lastly, the use of ML techniques that help create mechanisms to detect possible attacks or anomalies on the network. This framework is intended to provide a real-time analysis application in the institution ISCTE – Instituto Universitário de Lisboa (Iscte), offering a more complete, efficient, and secure monitoring of the data from the different devices comprising the network.

Keywords: Data monitoring, Anomaly detection, Attack detection, SIEM, Big Data, ML algorithms.

Resumo

A multidisciplinaridade das sociedades contemporâneas obriga-nos a perspetivar os sistemas informáticos como uma das maiores dádivas de que há memória. Todavia o seu incremento implica uma mandatária força de segurança para utilizadores, força essa em forma de ferramentas eficazes e robustas no combate ao cibercrime a que os utilizadores, individuais ou coletivos, são sujeitos quase diariamente. A monitorização e deteção deste tipo de problemas tem de ser assegurada em tempo real, permitindo assim, às empresas intervenções frutuosas, rápidas e em uníssono.

A framework proposta é alicerçada numa simbiose orgânica entre ferramentas open source credíveis, acessíveis pecuniariamente e eficazes na monitorização de dados, recorrendo a um sistema baseado em técnicas de *Security Information and Event Management (SIEM)*, *Big Data* e *Machine Learning (ML)* comumente aplicadas para a criação de sistemas de monitorização em tempo real. Dissecando esta framework, é composta pela metodologia *SIEM* que possibilita a monitorização de dados em tempo real e em simultâneo guardar a informação, com o objetivo de auxiliar as equipas de investigação forense. Em segundo lugar, a aplicação do conceito *Big Data* eficaz na manipulação e organização do fluxo dos dados. Por último, o uso de técnicas de *ML* que ajudam a criação de mecanismos de deteção de possíveis ataques ou anomalias na rede. Esta framework tem como objetivo uma aplicação de análise em tempo real na instituição ISCTE – Instituto Universitário de Lisboa (Iscte), apresentando uma monitorização mais completa, eficiente e segura dos dados dos diversos dispositivos presentes na mesma.

Palavras-chave: Monitorização de dados, Deteção de anomalias, Deteção de ataques, sistemas *SIEM*, *Big Data*, algoritmos *ML*.

Contents

Annexes

List of Figures

List of Tables

List of Listings

Acronyms

AE Auto Encoder 20, 21
ANN Artificial Neural Network 21

BN Bayes Network 19
bWAPP buggy web application 39

CNN Convolutional Neural Networks 20

DBN Deep Belief Network 20
DCNN Deep Convolutional Neural Networks 20, 21
DDoS Distributed Denial-of-Service 9
DL Deep Learning xix, 17, 20, 21
DNN Deep Neural Network 20
DoS Denial-of-Service 8, 9, 18, 48
DSRM Design Science Research Methodology 3
DTA Decision Table 19
DTREE Decision Tree 17, 20, 21

ELK Elastic, Logstash and Kibana 16, 23, 31, 37, 41
EM Expectation Maximization 18

FN false negative 19
FP false positive 18, 19

HDD Hard Disk Drive 39
HDFS Hadoop Distributed File System 15, 16, 23, 30, 34, 35, 36

IDS Intrusion Detection Systems xix, 1, 20, 21

IF IsolationForest 18
IOT Internet of Things 9
IP Internet Protocol 9, 10, 56
Iscte ISCTE – Instituto Universitário de Lisboa ix, xi, xvii, 2, 3, 4, 5, 7, 25, 39, 40, 43, 62, 63, 64
IT Information Technology ix, 1, 2

KNN K-Nearest Neighbor 17, 18, 19, 20, 21

LDA Latent Dirichlet Allocation xix, 45, 46
LLR Lasso Logistic Regression 18, 19
LR Logistic Regression 18, 19
LSTM Long short-term memory 12, 20, 21

MIB Management Information Base 9, 10
ML Machine Learning ix, xi, xix, 5, 17, 18, 19, 20, 21, 25, 27, 30, 31, 34, 39, 41, 43, 45, 46, 51, 54, 55, 62, 64
MLP Multi-Layer Perception 17, 18, 19, 20

NB Naïve Baeyesian 17, 18, 19, 20

OCSVM One Class Support Vector Machines 18
OID Object Identifier 9

QDA Quadratic Discriminant Analysis 20

RAM Random-access Memory 39
RBM Restricted Bolzmann Machines 20
RF Random Forest xvii, 9, 18, 19, 20, 34
RNN Recurrent Neural Networks 20, 21
RT Random Tree 19

SEM Security Event Management 1, 16
SIEM Security Information and Event Management ix, xi, 1, 2, 4, 7, 16, 25, 27, 31, 51, 61, 62
SIIC Serviços de Infraestruturas Informáticas e de Comunicações 2, 3, 39, 40, 43
SIM Security Information Management 1, 16
SNMP Simple Network Management Protocol xix, 9, 10
SVM Support Vector Machines 9, 17, 18, 19, 20

TP true positive 18, 19

UDP User Datagram Protocol 9, 10

VM Virtual Machine 39, 40, 41

1 Introduction

1.1 Motivation

With the proliferation of the internet in a wide variety of products, from a computer to a simple sensor, the topic of cybersecurity becomes increasingly discussed. There is a rise in the frequency of reports of organizations that have suffered from cyber-attacks. This issue influences the operation of devices and the trust of users in a corporation [4, 60].

It has been reported that attacks can only be detected by correlating events and logs from different security components [35]. Using a component operating individually, such as Intrusion Detection Systems (IDS) does not bring a holistic view to a system. Organizations need to put themselves in the attackers' heads and reflect on the different layers that can be affected, correlating them and making the system a more comprehensive and effective solution for detecting possible attacks [81].

Systems based on *Security Information and Event Management (SIEM)* methodology [52, 18] receives data from various security devices, normalizes into a common representation for correlation purposes. They collect data to a repository for analysis: provides *Security Information Management (SIM)* methodology, log-term storage, analysis and reporting for forensic needs and *Security Event Management (SEM)* methodology, real-time monitoring, correlation, notifications and console views [35] allowing the evaluation of various messages and alerts from different components of Information Technology (IT) which is useful for the development of systems to detect possible intrusions or anomalies.

There are several types of attacks. A very popular class are stealth attacks [26], which are often undetected by conventional attack detection systems and subtly consume resources of a device over time to fulfil their aim. For example, some attacks scan the data stored on the disks to discover sensitive information about the target. Therefore, it is relevant to study new ways of analysing networks by supplementing the classic approach of using network traffic data. To disseminate attacks, it is usually essential to exploit the components of the victim's device. Perform a system assessment to get a holistic view of the current behaviour, which may be affected by the attack.

For this reason, the exploration of techniques to collect and analyse component system

metrics data to enhance the detection of attacks is an interesting factor to evidence new ways of providing security systems within a network. The proposed framework extends the work carried out by Dias et al. [37] with the introducing of system metrics data. This data provides an insight into the health of the devices, allowing the analyser to act more quickly on the detection of abnormal behaviour that may indicate the presence of attacks not detected by conventional attack detection systems.

As predicted by several authors, such as by Chen [29], the increasing number of diverse devices on the Internet has created an exponential number of data traffic. Thus, it is important to create systems capable of collecting and ingesting a large amount of data. The topic *Big Data* arises with the purpose of presenting techniques capable of processing large amounts of data. The incorporation of *Big Data tools* in systems based on *SIEM* methodology has demonstrated several benefits, bringing a more distributed and efficient implementation [25].

We propose a framework based on *SIEM* methodology and *Big Data* tools to analyse network and system metrics data from different security components in a network in real-time. We developed on a *Plug-in-and-play* concept, that is, one where each individual component can be easily replaced and based on open-source tools. In our implementation, we used open-source tools studied in the related work to create independence from commercial tools.

The Serviços de Infraestruturas Informáticas e de Comunicações (SIIC), the IT department of ISCTE – Instituto Universitário de Lisboa (Iscte), have the objective of implementing a framework capable of monitoring possible attacks and anomalies in network systems in real-time. At the moment of this research, there was only a framework in place that verified the health of a device by its response times. This framework doesn't deliver many insights about the health of the network, since it only shows that there was a failure in the response time, and it is not possible to analyse what might have caused this problem.

1.2 Research Goals

To conduct this research, we formulate the following questions:

1. *What type of techniques for the development of a network analysis framework in real-time should be applied in this research?*

2. *Are there any advantages in implementing a network analysis framework for simultaneous attack and anomaly detection?*

3. *How to measure the results obtained in the proposed framework deployed on the Iscte network?*

4. *Are there benefits in the implementation of the proposed framework on the Iscte network?*

Question 1 establishes the basis of this research. First is necessary to identify what techniques should be applied to develop this type of framework. This is followed by Question 2 which will help appraise the advantages of having a framework that takes both attacks and anomalies into consideration. Question 3 intend to understand how to measure the results obtained in the application of the proposed framework on the Iscte network. We finalise with question 4 to discuss the benefits of the deployment on the Iscte network.

The goal of this work is by the end of the research conducted to be able to answer all the research questions proposed above. The aim is to understand the importance of network analysis systems, what are the best practices, how to implement them and evaluate the results. The objective is to apply the knowledge acquired during the research, implement an network analysis framework on the Iscte network. At the end, evaluate the results to understand if it has added benefit for the institution.

1.3 Methodology

The methodology of this research follows the Design Science Research Methodology (DSRM) which relies on a problem-solving paradigm [83]. It seeks to extend the boundaries by creating innovative artifacts focused on "utility", i.e., at the construction and evaluation of generic means and relations [99]. Design Research complies six activities [80]:

1. Problem identification and motivation
2. Define the objectives for a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

This work relies on a client/context-initiated solution, based on observing a practical solution that worked or as a result of a consulting experienced [80]. In this case, the DSRM process will start at activity 4, since the SIIC proposed the deployment of a network analysis system after observed other implementations in this area experimented and worked. As the solution suggests, we need to go back to previous activities to apply rigour.

We start with activity 1 where we define the research problem and the motivation of a solution. The problem will serve to produce an artefact that can provide a solution. This activity comprises the state of the art of the problem and the motivation for its solution. Once that is completed, we need to infer the objectives for a solution and what is feasible.

The next activity includes the design and development of the artefact. Afterwards, the artefact is deployed to solve one or more instances of the problem. After deployment, we need to measure how well the artefact provides a solution to the problem. This activity can be a comparison of the artefact's functionality with the objectives of the solution. If the results are not as desired, we may go back to activity 2 and reformulate the objectives of the solution and perform a new iteration of the research. When the artefact achieves the expected, the results should be communicated.

For this dissertation, Section 1.1 defines the motivation for a solution. The research questions established in the previous section outline the research problems and the objectives for a solution. The network analysis system introduced on the Iscte network will be considered the artefact. Research questions 4, 5, 6 aim to evaluate the results. After achieving the expected, the results will be communicated.

The research started with keywords like *"SIEM"*, *"Network Analysis"*, *"Anomaly detection"*, *"Attack detection"*, and *"Zero-day attacks detection"*. This initial search aimed at identifying the current systems and tests carried out on them due to the non-existence knowledge on the subject. From this preliminary review, it was possible to identify the essential components for the development of a network analysis framework. Following this step, the investigation was divided into topics.

Firstly, the different types of data used in network analysis were identified. Afterwards, the algorithms applied in these scenarios were reviewed, comparing the results achieved in the different types of algorithms. Once these topics were studied, the current tools employed for the building of these systems, such as, for example, systems based on *SIEM* methodology, were also discussed. Bearing the idea that it is important to create a framework with the ability to process large amounts of data if necessary, research on the application of "Big Data"tools in analysis systems was carried out to provide a scalable artefact. Last but not least, it is very pertinent that the system offers a user-friendly interface, creating a simple but informative visualisation that delivers a good user experience(Table 1.1 describes the Literature review divided by the different topics).

To achieve the objectives of this research, *Google Scholar* and *Mendeley* were used as search engines. Recent works published in *IEEE*, *ACM* and *Springer* were given greater importance. However, while reading a paper, it may arise the interest to read a referenced paper, which makes the age of the papers increase. At the end of the research, the window of articles ranges from 2021 to 2005 (Table 1.2 depicts the Literature review divided by range of years).

1.4 Main Contributions

This thesis provided the following contributions:

- The creation of a *Plug-in-and-play* framework, based on data streaming, that is, a network analysis framework working in real-time, from edge to edge, where the

Table 1.1: Literature Review per topic.

Topics	Number of documents
Network Analysis Frameworks	5
Data Collection	41
Big Data Toolkits	8
Network Analysis Techniques	15
Data Visualization	1
Total	70

Table 1.2: Literature Review per date.

Date	Number of documents
2021-2018	42
2017-2014	21
2013-2010	5
2009-2005	2
Total	70

result of Machine Learning (ML) models are visualised on a dashboard.

- The deployment of a network analysis framework on the Iscte network.
- Extension of the work developed by Dias et al. [37] with the addition of the importance of metrics data in the development of a robust network analysis framework.
- A scientific article to be published in a conference.

1.5 Thesis Structure

The dissertation is divided into 6 chapters. Chapter 1 introduces the thesis by highlighting the motivation, the questions, and the contributions of the research. Chapter 2 investigates the work carried out in the development of a network analysis framework. Chapter 3 starts by making a short motivation to the proposed Framework. During the chapter, it is described how the framework was designed. In chapter 4 the framework proposed is implemented. Chapter 5 discuss the results of some experiments in order to evaluate the deployment. Chapter 6 presents the conclusions of the work conducted in this thesis and the future work suggested.

2

Related Work

As mentioned in Chapter 1, we pursuit two intertwined goals: (1) to implement a network analysis and anomaly detection system and (2) to use open-source tools in the ISCTE – Instituto Universitário de Lisboa (Iscte) network but targeting mainly anomalies and attacks.

In this chapter, we review work that relates to those goals. Hence, Section 2.1 introduces work on collection of data of interest, such as databases used in certain applications and use cases, as well as existing tools for the purpose.

Then, Section 2.2 refers to Big Data tools and the idea of implementing a system based on Security Information and Event Management (SIEM) methodology. In Section 2.4 it is presented some analysis techniques but different algorithms, in particular the assessment of their suitability to the problem we are dealing with.

Then, Section 2.5 discusses the importance of data visualisation for this type of systems, alongside some tools of interest. Finally, Section 2.6 presents some frameworks that are available for different types of environments.

2.1 Data Collection

Not surprisingly, the data collected from network systems is critical to understand and analyze its own functioning. Indeed, discovery of behaviors and health of the systems are paramount. In that respect, there are different features to be considered: about flow, content and time, the basic plus the generated ones, as well as labeling [67]. Also, the protocols to be deployed in the network are important to gather the crucial data, and also the tools we may have at our disposal helping us to carry out the tasks.

In the following sections we will discuss these issues.

2.1.1 Attack Detection Data Features

Considering that data is being collected over time, when an attack occurs on a network, it is interesting to explore a possible association between this attack and the data collected in order to recognise the potential influence of that type of attack on the data.

For example, an Denial-of-Service (DoS) attack causes too many requests to a server in a short period of time, which causes the server to become overloaded and no longer have services available to respond to the rest of the clients. From data collected, it is possible to verify that this type of attack influences server response times. Like this example, all attacks influence certain data.

This allows the creation of datasets that represent how different attacks can influence the values of the network data. From these datasets, it is possible to start developing several studies in order to identify possible attacks.

Table 2.1: Datasets used in previous research.

Paper	Dataset	Year
[48]	CICIDS2017 and NSL-KDD	2020
[84]	CIDD (DoS attacks)	2020
[42]	CSE-CIC-IDS-2018	2020
[17]	CSE-CIC-IDS-2018	2020
[2]	Kaggle Website	2019
[86]	NSL-KDD	2019
[56]	KDD Cup'99 and NSL-KDD	2018
[46]	NSL-KDD and UNSW-NB15	2018
[87]	KDD Cup'99 and NSL-KDD	2018
[41]	KDD Cup'99	2018
[65]	Self-collected	2018
[34]	KDD Cup'99	2018
[74]	NSL-KDD	2016
[1]	NSL-KDD	2015

From this idea, several databases have emerged:

- KDD99 dataset was established as dataset standard used on attack detection studies. This dataset includes a diverse number of attack simulations on a military network environment.

- NSL-KDD dataset is based on KDD99, was created by DARPA within 7 weeks (compressed raw TCP dump). This dataset contains nearly 4.900.000 single connection connections and 43 features [36].

- UNSW-NB15 dataset was created by Cyber Range Lab do Australian Centre for Cyber Security (ACCS). This dataset contains 257.673 network traffic records and 49 features. Was used the IXIA tool to generate benign and attack traffic and Bro-IDS and Argus tools for monitoring the traffic [68].

- CICIDS2017 dataset contains benign and malign updated traffic, similar to real network flow (PCAPs). This dataset includes network analysis, transforming PCAPs flow in CSV instances. The search is within 5 days [57].

- CICIDS2018 [42] is composed of 50 attack machines and the victim structure with 5 departments with 420 machines and 30 servers. Includes the capture of network traffic and 80 features. The Database consists of two classes of profiles:

  - B Profiles – Encapsulate user entity behaviors using various machine learning and statistical analysis techniques (such as K-Means, Random Forest (RF), Support Vector Machines (SVM), and J48). Protocols simulated in the test environment: HTTPS, HTTP, SMTP, POP3, IMAP, SSH, and FTP. "CIC-BenignGenerator" is the tool to generate benign traffic.
  - M profiles – attempt to describe an attack scenario. In the simplest case, humans can interpret these profiles and then execute them. Ideally, standalone agents along with compilers would be employed to interpret and execute these scenarios. It is composed of seven different scenarios: Botnet, Brute-Force, DoS, Distributed Denial-of-Service (DDoS), Heartbleed, Web attacks, and Infiltration of the network from inside.

2.1.1.1 Anomaly Detection Protocols

The process of collecting data to detect anomalies requires that devices in the network are properly managed. By devices we mean cable modems, routers, switches, servers, workstations, printers, Internet of Things (IOT), etc. In that respect, the Simple Network Management Protocol (SNMP) is the standard protocol to manage those devices.

We also need protocols to transmit log messages over Internet Protocol (IP) networks. Hereafter we describe some of those protocols.

SNMP [64]. The communication is done through a manager that has the role of monitoring a group of agents. An agent is characterised by a device in which the SNMP protocol is active whose function is to send information to the manager. This agent can interact with its different objects through the Object Identifier (OID) . Each OID identifies a unique object that can be read or changed via SNMP. The different OID of an agent are stored in a Management Information Base (MIB) file that organises the data in a hierarchical tree structure.

SNMP operates at the application layer, messages are transported via User Datagram Protocol (UDP). The protocol allows active management of tasks such as configuration changes, via remote modification of variables.

However, there are several disadvantages [89]. For example:

- It is possible to use SNMP to attack the network. Can be used to penetrate the network. A significant number of software tools can scan the entire network using SNMP and from there discover flaws in the read-write mode configuration, being susceptible to attacks. It is important to be careful to configure which IP addresses

are accepted for messages. However IP address spoofing (pretends to be another IP source) remains a security concern.

- SNMP authentication: security depends on the SNMP version. SNMP is not being fully used by several systems, because most of the devices have the SNMPv3 version which is the only one that guarantees security and several devices are not able to be configured via individual MIB.
- SNMP Autodiscovery: Many SNMP implementations include a type of autodiscovery where a new network component, such as a switch or router, is automatically discovered and grouped.
- There can be correlation problems when joining information from different devices that may not have the same indexing scheme.

Several papers study the use of SNMP data to detect network anomalies (See Table 2.2). The results shows that Interface and IP groups are the most affected by attacks.

Table 2.2: Research on SNMP messages.

Paper	Dataset	Year
[72]	Interface, IP, TCP, UDP e ICMP groups	2018
[73]	Interface group	2018
[5]	Interface, IP, TCP, UDP e ICMP groups	2019
[63]	IP group	2019
[71]	Interface, IP, TCP, UDP e ICMP groups	2019

Syslog. This protocol is used to transmit log messages over IP networks. The server allows the log information of all network devices to be centralized on a single machine. The information is usually sent via UDP. It is possible to manage, search and archive all message logs.

Log information is very important for troubleshooting and for further analysis since network devices have a low memory capacity, which causes device history to be erased after a certain time.

A log message is made up of several parts, as shown in Figure 2.1.

Figure 2.1: Structure of a Syslog message.

In that respect, the meaning of the parts are:

- The timestamp indicates when the message occurred.
- The facility indicates the origin of the message, represents the process that generated the message.
- The severity indicates how urgent the log message is.
- The mnemonic which is the code to identify the message.
- The description that describes the log message.

Table 2.3: Syslog facilities.

Code	Keyword	Description
0	kern	Kernel messages
1	user	User-level messages
2	mail	Mail system
3	daemon	System deamons
4	auth	Security/authorization messages
5	syslog	Messages generated internally by syslogd
6	lpr	Line printer subsystem
7	news	Network news subsystem
8	uucp	UUCP subsystem
9	cron	Clock daemon
10	authpriv	Security/authorization messages
11	ftp	FTP daemon
12	ntp	NTP subsystem
13	security	Log audit
14	console	Log alert
15	solaris-cron	Clock daemon
16-23	local	Local use

Table 2.4: Syslog severity.

Code	Severity	Description
0	Emergency	System is unusable
1	Alert	Action must be taken immediately
2	Critical	Critical conditions
3	Error	Error conditions
4	Warning	Warning conditions
5	Notice	Normal but significant conditions
6	Informational	Informational messages
7	Debug	Debug-level messages

The most relevant parts of a log message are facility and severity. It can be defined on network devices that only send messages with a certain severity, in order to avoid overloading with irrelevant messages.

Several researches studied the usefulness and efficiency of Syslog messages for detecting anomalies in the network. (Described on Table 2.5)

He et al. [47] presents a detailed review and evaluation of six state-of-the-art: log-based anomaly detection methods, including three supervised methods and three unsupervised methods and also presents an open-source toolkit allowing ease of reuse. Baseman et al. [13] study an anomaly detection framework that combines graph analysis, relational learning, and kernel density estimation to detect unusual Syslog messages. Du et al. [38] approach "DeepLog", a log-based anomaly detection using Long short-term memory (LSTM) to model a system log as a natural language sequence. This allows automation of learn log patterns from normal execution and detect anomalies when log patterns deviate from the model trained. It is possible to update incrementally in an online fashion to adapt to new log patterns over time. Wang et al. [96] presents combination of feature extraction methods from natural language processing (Word2vec, TF-IDF) and anomaly detection methods from deep learning (LSTM). Vaarandi et al. [95] presents a novel data-mining based framework for detecting anomalous log messages from Syslog: patterns from the last N days and W weeks (so called mining windows) are then used for creating rules that match messages reflecting normal system activity. Nourtel et al. [75] approach an unsupervised deep degenerative model, after trained on a sufficiently large corpus, the generative model shall capture the "normal" behaviour of the system, and deviations from these predicted logs may be tagged as anomalies.

Table 2.5: Scientific research on Syslog messages.

Paper	Dataset	Year
[47]	HDFS log dataset from Amazon EC2 platform. BGL log dataset recorded by the BlueGene/L supercomputer.	2016
[13]	Syslog messages from a VM. Syslog messages from trinity supercomputer.	2016
[38]	HDFS log dataset from 200 Amazon's EC2 nodes. OpenStack log dataset with one control node, one network node and eight compute nodes.	2017
[96]	Thunderbird supercomputer. ALERT, FATAL and FAILED as anomalies.	2018
[95]	OS level Syslog messages from 543 Linux servers.	2018
[75]	Bull-ATOS HPC logs files dataset from Deutsches Klimarechenzentrum Supercomputer.	2019

2.1.2 Tools

In real environments, tools are essential as they provide flexibility in the types of data to be collected, and according to the needs of the organization. Among many publicly available, three tools should be pointed out: *Wireshark*, *CICFlowMeter* and *Beats*.

Wireshark

It is the most popular tool used as a network protocol analyser (online and offline) [100]. It allows an extensive analysis of what is happening on the network using different protocols. Runs on multiple platforms. It can be used in a GUI or by the command line (Tshark). Allows reading and writing to different types of files. Decryption is available with different protocols. It has an intuitive analysis.

CICFlowMeter

It is the network traffic flow generator and analyser used by the CICIDS2018 dataset [30]. The tool takes network capture files (such as Wireshark generated files) as input and generates bidirectional flow metrics where the first packet of each flow represents forward and backward. 80 statical network features such as Duration, Number of packets, Number of bytes, Length of packets, etc. are generated. The evaluation depends on the protocol to be used: A TCP connection, the end of the flow is considered when a FIN packet is received, if it is a UDP connection, the timeout will indicate the end of the flow.

Beats

It is a free and open-source ecosystem of data agents. Allows automation of data collection [14]. It works as an agent installed on a particular machine that has the function of analysing: files (Filebeat), metrics (Metricbeat), network data (Packetbeat), logs (Winlogbeat, Auditbeat), availability (Heartbeat) and cloud services (Functionbeat). Beats accept many types of log formats from such widespread systems as MySQL, Apache, NGINX, etc. The data collected by Beats can be sent to diverse systems.

2.2 Big Data Toolkits

The data under consideration is generated in real-time so clearly it fits in the so-called big data category. The volume is big, that is, there are large scale data sets to deal with and therefore it is quite challenging to analyse them. Other characteristics to consider in the theme of big data are speed, variety, value, and veracity. And recall that the human brain is not really good at making assumptions about huge amounts of data [6].

That being said, fortunately, several tools have emerged. Some of them will be introduced below.

2.2.1 Apache Kafka

It is an open-source distributed event streaming platform used to process data pipelines, streaming analysis data integration, and mission-critical applications [55]. Kafka is used by many enterprises in different areas.

Kafka's structure is described on Figure 2.2 lies on a cluster distributed by n brokers, each broker works as a server that has its own local storage. Kafka consists of Producers who produce data for Kafka and Consumers who consume data from Kafka. The Producer and the Consumer are decoupled, i.e., they do not know of each other's existence.

The strong point of Kafka is to be composed of several topics. Topics are sets of the collection of events/messages with similar characteristics. For example, a set of information from twitters and another set of information from viewing videos on YouTube.

A topic has a structure identical to a logs file, it consists of a queue of messages over time. Every time it is received a new message, it is placed at the end of the queue.

Topics can be divided into several partitions that can be spread over several brokers, which allows for a distribution of information for faster processing.

Each Consumer / Producer can link to a topic. Being a queue system there can be n Producers connected to the same topic.

Consumers only read the information, i.e., the information remains in the partition even after being read, which allows the existence of n Consumers. Kafka is responsible for keeping the offset of each consumer. The time Kafka stores messages can be adjusted.

To prevent the possible failure of a system, it is possible to replicate the information. It is possible to have a leader partition and secondary partitions distributed by other brokers that will safeguard the information of the leader partition.

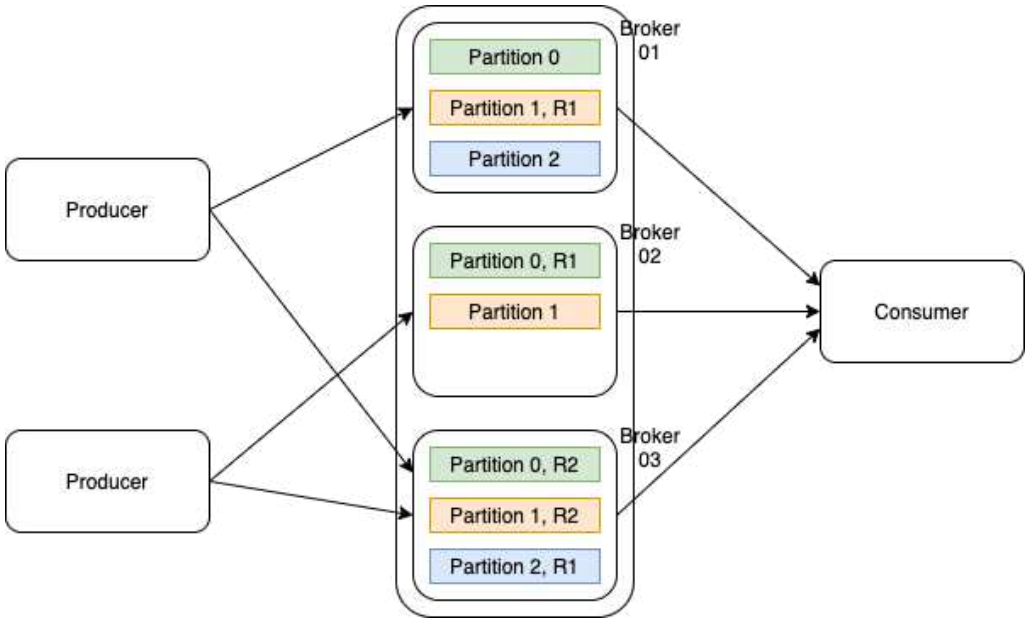

Figure 2.2: Apache Kafka's architecture.

2.2.2 Apache Spark

It is a data processing tool [33]. Spark distributes tasks across several nodes. It is a driver program that runs the main function and distributes the tasks across the different nodes described on Figure 2.3. It is possible to work in batch or real time. It was created due to the limitations of map reduce which takes time to process, it is not suitable for operations such as filtering, it is not made for large data because it takes time to process, and it does not fit in iterative executions like k-means that need to process the data several times.

Apache Spark addresses all the limitations of map reduce, it's about 100x faster on memory and 10x faster on disk which is best suited for big data and machine learning. Is composed by several models such as: Spark core and RDDS, Spark SQL, Spark Streaming, ML Lib, Graph X.

Advantages of Apache Spark:

- Speed – extends the map reduce model to support computation such as stream computation and interactive queries.
- Programming languages – programming can be done in several languages such as Java, Python, Scala, etc.
- Hadoop support – it is possible to integrate Spark with Hadoop, so allowing to create distributed datasets with Hadoop Distributed File System (HDFS) files.

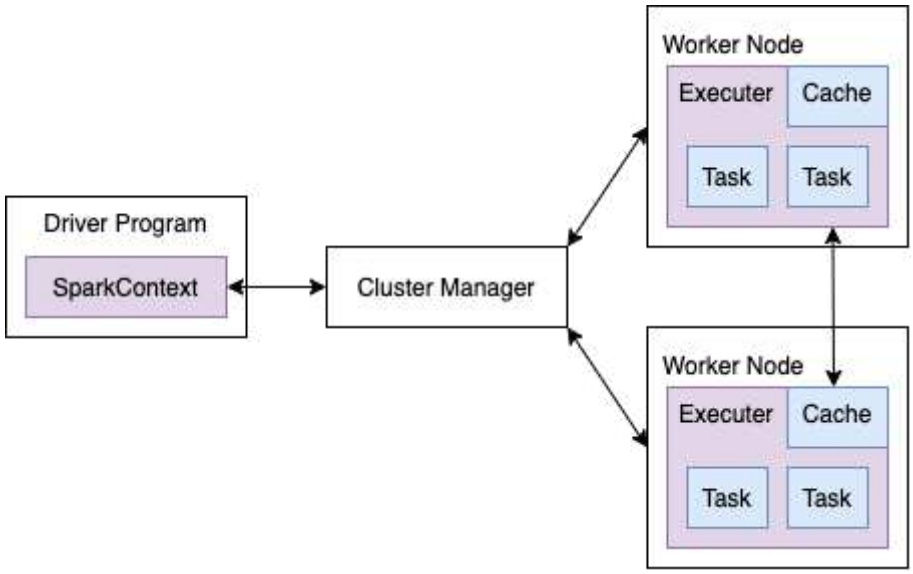

Figure 2.3: Apache Spark's architecture.

2.2.3 Hadoop Distributed File System

It is an open-source framework for large-scale data processing. Hadoop has a cluster architecture. It works like a set of machines that work together. Allows for high availability,

computational load balancing, and parallel processing. Its architecture offers horizontal scalability, that is, a set of several small server nodes.

HDFS [88] is a Hadoop's distributed storage system that deals with subjects such as volume, speed, and variety of data, creating a scalable system. Data is distributed over several servers. Hadoop divides files into blocks and distributes them across nodes. To be fault-tolerant, blocks are replicated so that they are not just on one server.

2.3 SIEM

SIEM methodology is the state-of-practice used on network security that allow *a holistic view of the security management* [85]. SIEM is a combination of Security Information Management (SIM), which provides long-term storage, analysis and reporting of log data, and Security Event Management (SEM), real-time monitoring, correlation of events, notifications, and console views. The purpose of SIEM is to allow the creation of an organized and proactive network system in that it implements techniques that allow the evaluation of data in real time but also important forensic research to disassemble a problem. Besides these factors there is the possibility of adding proactive reactions when problems are identified such as alerts when something is wrong.

Systems based on this methodology provide collection, normalization, and evaluation of messages from different data sources. Their capability to correlate data from different source provides a meaningful data to analyse anomaly patterns.

A system base on SIEM methodology uses data from end user devices, network devices, servers, firewalls and intrusion detection and prevention systems. The data is forwarded to a central unit that normalize and correlate the data. Annually, Gartner [54] provide the magic quadrant from SIEM, where different systems based on SIEM methodology are meticulously evaluated for their qualities and cautions. Among the entire report, Splunk and Elastic, Logstash and Kibana (ELK) stand out.

2.3.1 Splunk

It is a set of products for Enterprise and Cloud supported by Splunk Search Processing Language [78], provide searching, correlation, visualization, and alerting. It can show alerts from an in-depth device perspective, has efficiency in sort and filter data, big data architecture. It's an all-in-on product, fast installation, strong visualization, flexible searches, and reports.

However, the free version being limited with only 500MB per day, which makes it impossible to use this version to analyse the network of an organisation.

2.3.2 ELK

The combination of Elasticsearch, Logstash and Kibana [39], free and open source, provide an engine for easily storing data in a cloud or in the server. Uses Apache Lucene

database to index, can retreat much faster. It can filter, process, correlate and generally enhance any log data that it encounters. It has an UI with reports, analytics, configure stats, configure data, reports, dashboards and so on. It has the possibility of integrating machine learning algorithms to analyse the data.

2.4 Network Analysis Techniques

As we witness more and more network attacks, it is challenging to identify such attacks. The analysis can be divided into three different types [102], as follows:

- **Misuse-based**, which consists on detecting known attacks by using signatures. It does not have many false alarms. However, since the rules are fixed, zero-day attacks can't be detected.
- **Anomaly-based**, where it identifies anomalies as deviations from normal behavior. It has the capacity to zero-day attack detection. The results can then be used to create new signatures on Misuse-based.
- **Hybrid**, a combination of misuse and anomaly detection. Most of Machine Learning (ML) and Deep Learning Deep Learning (DL) methods are categorized as hybrids.

As a large portion of analysis relies on ML and DL methods to some extent, in the following sections we discuss their underlying concepts.

2.4.1 Machine Learning

The critical aspect of ML is to generate predictions automatically. In general, the process starts by choosing features engineering, an algorithm to process data, using a training dataset and to evaluate the model itself. Then, it follows the prediction of unknown data, that is, the classification of data of interest, based on the using the trained models previously built. There are three major approaches [102, 3, 19]:

- **Supervised learning**, that relies on training via labeled data. Typically, it learns a function that predicts if an output from the selected features is considered a normal or an anomaly class. The training data has usually fewer attack instances compared to normal instances, which may disrupt this approach. There are several algorithms available like SVM, K-Nearest Neighbor (KNN), Decision Tree (DTREE), Naïve Baeyesian (NB) and Multi-Layer Perception (MLP).
- **Semi-supervised**, which trains the data assuming that only normal classes have labelled instances, so reducing efforts and achieving high accuracy. It learns a model for normal behaviour and use it to identify anomalies.

- **Unsupervised learning**, which does not require label data. It assumes that normal instances are much larger than anomaly instances and the algorithm discovers patterns in data to create rules. There are several algorithms available such as clustering, which splits similar data into groups. Furthermore, clustering can be divided in two groups, regular clustering (row cluster) and co-clustering (row and columns cluster).

Belavagi & Muniyal [15] evaluate the performance of supervised ML classification algorithms, Logistic Regression (LR), Gaussian NB, SVM and RF on NSL-KDD dataset for intrusion detection. As for evaluation, it is used true positive (TP) rate, false positive (FP) rate, precision, recall, F1-score and accuracy. Experimental results show that RF outperforms and SVM under performs (due to large number of features).

Several clustering algorithms have evolved over the time. Ahmed et al. [3] presented an evaluation table with different clustering algorithms: k-means, improved k-means, k-medoids, Expectation Maximization (EM) clustering, distance-based anomaly detection from the investigation of Syarif et al. [92]. Distance-based anomaly detection reached 80,15% of accuracy, while the straightforward k-means clustering has 57,81% of accuracy. The authors also mention the benefits of using co-clustering for DoS attacks. It is compared the purity (percent of the total number of data points that were classified correctly) of normal and attack instances of Ahmed and Mahmood (2014b) [10] approach (co-clustering detection for DoS Attacks) and Papalexakis et al. [77] approach (co-clustering detection for all types of network attacks). The results show an improvement on Ahmed and Mahmood (2014b) approach.

Arunraj et al. [9] presented a comparison between supervised (LR and Lasso Logistic Regression (LLR)), semi-supervised (One Class Support Vector Machines (OCSVM)) and unsupervised learning (IsolationForest (IF)). In the test training period, LLR is faster than LR without sacrificing much of its performance. IF performance is poorer regarding OCSVM performance but has a better computational time during test. Although the main challenge of an algorithm to anomaly detection is to minimize as good as possible false negatives, which is when the anomaly was not detected. The results show that OCSVM algorithm has a low number of false negatives while it has an acceptable number of false positives. The experiment also shows that when the test contains no unknown data, supervised outperform unsupervised learning. However, when the test contains unknown data, the performance of unsupervised will not degraded whereas supervised will.

Zamani & Movahedi [105] approach the Laskov et al. [61] experimental framework of supervised (KNN, MLP, SVM) and unsupervised (y-algorithm, k-means and linkage clustering) learning techniques comparison. The test was divided in two scenarios: firstly, the train and test data come from the same unknown distribution; secondly, the test data came from a new malicious pattern. The results showed that in the first scenario, supervised algorithms have a better accuracy (decision tree reaching 95% TP and 1% FP).

However, in the second scenario, the performance of supervised algorithms decreased significantly while unsupervised algorithms were weakly affected. The experiment demonstrates that usually unsupervised algorithms have better performance than supervised algorithms.

Almseidin et al. [8] analyses the performance of most common ML methods (J48, RF, Random Tree (RT), Decision Table (DTA), MLP, NB and Bayes Network (BN)) based on KDD dataset. The evaluation is focus on false negative (FN) and FP metrics. The results have showed that there is no single perfect ML algorithm. RT algorithm has achieved the higher FN rate of 0.093, meaning that a large number of attacks are classified as normal instance. On other way, DTA algorithm achieved the lowest FN rate of 0.002 but also achieved the FN rate of 0.073. Therefore, there is no balance between FN rate and FP rate. So high accuracy rates are not enough to create a good intrusion detection approach.

Table 2.6 briefly summarizes the ML algorithms described above.

Table 2.6: Brief comparison of ML algorithms.

Paper	ML methods	ML metrics	Observations
[15]	Supervised: LR, NB, SVM and RF	TP rate, FP rate, Precision, Recall, F1-score and Accuracy	RFC outperforms. SVM underperforms due the fact of the large number of features.
[9]	Supervised: LR, LLR Semi-supervised: OCSVM Unsupervised: Isolation Forest	FN	OCSVM has a low number of FP while has an acceptable number of false positives. When the test contains no unknown data, supervised outperform unsupervised learning. When the test contains unknown data, the performance of unsupervised will not degraded while supervised will.
[105]	Supervised: (KNN, MLP, SVM) Unsupervised (y-algorithm, k-means and linkage clustering)	TP, FP	Unsupervised: algorithms have better performance than supervised algorithms.
[8]	J48, RF, RT, DTA, MLP, NB and BN	FN, FP	There is no single perfect ML algorithm. There is no balance between FN rate and FP rate.

2.4.2 Deep Learning

DL is a branch of ML that *mimics the human brain* [102], and it is composed by linear and non-linear transformations. DL focus on classification and learning multiple layers. The major differences between DL and ML are described in Table 2.7.

Table 2.7: Major differences between DL and ML.

	DL	**ML**
Data dependencies	Amount of data	Don't
Hardware dependencies	Optimized performance with GPU	Don't
Feature processing	Don't	Depends on the accuracy of features
Problem solving method	End-to-end	Subset the problem
Execution time (train)	Takes time (weeks)	Seconds to hours
Execution time (test)	Stay stable	Time was affected by the amount of data
Interpretability	Results are difficult to explain	Clear rules

The most used DL algorithms are Restricted Bolzmann Machines (RBM), Deep Belief Network (DBN), Deep Neural Network (DNN), Auto Encoder (AE), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) [3, 59].

Naseer et al. [70] present a research about suitability of DL approaches for anomaly-based intrusion detection system, such as LSTM, CNN, AE and RNN. For the study, they have used ML methods like SVM, KNN, DTREE, RF, Quadratic Discriminant Analysis (QDA) and Extreme Learning Machine. The test is divided in two datasets, NSLKDDTest+ and NSLKDDTest21 (that contains unknown attack instances for test phase). The evaluation is done via classification metrics, area under RoC curve, accuracy, precision-recall, mean average precision, as well as test and training timings. From the results, Deep Convolutional Neural Networks (DCNN) is the best algorithm in AuC an mAP metrics, as shown in Tables 2.8 and 2.9. As for training and time, Decision-tree is the best algorithm. DCNN model took 109 seconds for training, two seconds for NSLKDDTest+ and one second for NSLKDDTest21 dataset. Therefore, the results show that DL is a promising methodology for network security.

Yin et al.[103] proposed a DL approach for an IDS (RNN-IDS). This study was performed with most common ML learning methods such as J48, NB, RF, MLP, SVM based on NSL-KDD dataset. The testing was carried out in binary and multi-class classification.

As for binary classification, RNN-IDS have been mapped with different hidden nodes

Table 2.8: Top five area under RoC curve results of models in relation to NSLKDDplus and NSLKDD21 datasets (Table from [70]).

Model Name	AuC for NSLKDDPlus	Model Name	AuC for NSLKDD21
DCNN	0.955	DCNN	0.916
LSTM	0.953	Convolutional AE	0.894
AE	0.937	Contractive AE	0.891
DTREE	0.937	DTREE	0.860
Contractive AE	0.920	KNN	0.825

Table 2.9: Top 6 mean average precision results from IDS models (Table from [70]).

Model Name	maP for NSLKDDPlus	Model Name	maP for NSLKDD21
DCNN	0.97	DCNN	0.98
LSTM	0.97	LSTM	0.97
DTREE	0.96	Convolutional AE	0.97
Contractive AE	0.95	Contractive AE	0.97
KNN	0.95	KNN	0.96
AE	0.95	DTREE	0.95

and learning rate. In the experiment, the higher accuracy (83.28%) is achieved by 80 hidden nodes and 0.1 learning rate. And DL approach outperforms regarding ML methods.

On the other hand, in multi-class (five) classification, RNN-IDS was mapped with 80 hidden nodes and 0.5 learning rate and it outperforms again with 81,29% accuracy, regarding ML methods. RNN-IDS also have a better performance than Artificial Neural Network (ANN) approach [51] which offers an accuracy of 79.9%. In addition, the authors have compared RNN-IDS with reduced-size RNN method applied to the KDD CUP 1999 dataset [51]. The results have shown an accuracy of 97,09% for RNN-IDS and an accuracy of 94,1% for the reduced-size RNN method. But we should notice that, without GPU, RNN-IDS takes 382 seconds more. Therefore, this highlights the growing importance of DL methods for both binary and multi-class classification.

2.5 Data Visualization

Data visualization is a powerful and simple way to get insight from complex and diverse data. As expected, various data visualization tools have emerged overtime to keep up with the growing amount of data collected. These tools have been constantly evolving and improving their ability to analyse a wide variety of data, even large scale data set. One of the features is to figure out interesting patterns and correlations in the data.

It is important to achieve a balanced and accurate representation of the data under analysis. In that respect, many aspects have to be taken into account. For example, visual noise, loss of information, large image perception, high rate of image changing, high-performance requirements, etc. [44]

On the other hand, big data itself has been a challenge for visualization, mainly due to scalability, functionality, and response time aspects. In order to face these challenges, there are methods and toolkits that have been used to deal with data processing and visualization all together.

One of the concerning aspects in a visualization tool is providing interactive visualization, that is, allowing the user to interact with the visual representation (image), which is successively adapting itself to the user's actions. (zooming, filtering, etc.)

In the case of attack and anomaly analysis systems, visualization allows the user a greater immersion and perception of what may have caused the detection of a problem. The images that have been created should provide the user both an overview and a useful detailed analysis, on-demand. At this point, the tools worth mentioning are the following:

Tableau

Tableau is one of the most popular tools used in business intelligence. It has user-friendly drag-and-drop functionality that allows any type of user a simple way to analyse data. It's fast and flexible, has a wide variety of charts. Supports multiple data sources. For simple tasks, it doesn't need programming skills but for more complex tasks it needs knowledge of the R language.

Power BI

Power BI is a data analysis service created by Microsoft that allows creating an interactive visualization via a simple and intuitive interface, so users can create their own analyses. It is also possible to use natural language for queries, therefore reducing the need for programming. Still, one can use various programming languages like R and different types of data sources.

Kibana

Kibana is an open-source tool that allows to create data visualizations from Elasticsearch data. Besides the classic charts available, it is possible to uses different types of data and to build a dashboard using natural language. Users are entitled to select how data is visualized in a interactive mode.

Plotly

Plotly, also known as Plot.ly, is built using the programming language Python and Django, a free framework suitable for creating web applications. Again, there is a large variety of charts available and it can be used with permissions for data analysis with greater privacy. The toolkit is available for Python, R, MATLAB and Julia APIs.

2.6 Network Analysis Frameworks

Considering the different aspects concerning network security mentioned in previous sections, it is important to build a framework capable of delivering a anomaly and attack detection solution for an organisation. In that respect, there are some relevant work that has been published. For example:

Babu et al. [11] presents the monitoring of logs generated by the Linux system and network logs configured in a firewall together with the ELK tool. The Syslog-ng collects all data, and it is sent to Filebeat, which works as an agent sending the collected data to Elasticsearch (repository). The Kibana tool is implemented for visualization. The collected sample is then analyzed in a controlled and isolated environment to determine its behavior and actions taken in the infected system.

Harikanth and Rajarajeswari [45] proposes the collection of logs from a server connected with endpoints. The Beats will send the log to Logstash and the output is given to Elasticsearch. Logs are forwarded to Threat Intelligence, where logs are prioritized from low to high. This classification is done by correlating the logs received in real-time with the threat classification based on the threat intelligence trained by the sources.

Almohannadi et al. [7] collect over 500 MB of Honeypot for one year via an Amazon Web Services (AWS) cloud. Two Honeypots called Kippo and Dionea were created, where Kippo is a low-high and Dionea is a medium honeypot. Both Honeypots appear as real operating systems, which attracts many hackers. Events are logged if someone tries to interact with the honeypot. Then they were then forwarded to the ELK tool for further analysis.

Teeraratchakarn and Limpiyakorn [93] create a virtual environment with Honeypot to catch hackers. It is configured to look like a real system with vulnerabilities. The difference is that it is isolated from the rest of the internet but monitored. It is evaluated every hour if there was a new registration. The registration is in the format of logs to discover attack patterns, which are discovered by rules created in scripts. Also, the implemented VirusTotal, a free service for analyzing files or URL addresses, is used for an extra malware assessment.

Dias et al. [37] presents an approach for network data analysis without training data in real-time. The approach uses clustering techniques to group hosts with similar behavior. Genetic zoom is introduced to identify the features that are most relevant to build a cluster. In order to detect stealth attacks, time stretching is implemented. For the testing phase, it is used a dataset from Los Alamos National Laboratory (LANL) corporate network and a dataset obtained from a large military infrastructure. As for deployment, big data toolkits like Apache Kafka, Apache Spark and HDFS are used.

2.7 Summary

This chapter presents a review of the state of the art focused on the topic in exercise. It is subdivided into the different fundamental components in the development of the final product. It begins by ascertaining what types of data are categorically used in network security applications and what techniques are available to collect them. Next, some of the various technologies used to apply the Big Data methodology are listed. Within these, the most celebrated researches based on more recent studies have been surveyed.

Following this, the application of different types of machine learning algorithms was investigated for the implementation of mechanisms for the detection of attacks and anomalies, with the goal of a representation of the results obtained using, then, visualization tools. Lastly, it was researched what kind of tools exist in the market with this same purpose. In brief, different Network Analysis Frameworks published and referring to the work we performed are evaluated.

3

Proposed Framework

This chapter introduces the proposed framework, starting with a motivation for its implementation and usage. Then it provides a more detailed explanation of its architecture, including how it should be deployed and used. The architecture follows a standard approach, partially mirroring the widely-used process CRISP-DM [28], but taken into consideration that it should lead to a data streaming application. Indeed, its main purpose is to analyse data in real-time. It also includes data modelling based on Machine Learning (ML) concepts.

At its core, there are four modules – *Data Collection*, *Data Ingestion*, *Data Modelling* and *Data Visualization*. Data flows through the modules, with processing alongside that might change its nature (derived, filtered, enhanced, etc.). Therefore, the architecture usability as far as data is concerned resembles a dataflow pipeline. The specificity of each module is explained below, as well the rationale behind the choices that were made.

In respect to data, there are two main categories of data to be processed and analysed: network data and system metrics data. Each one of these categories yields to specific mechanics of implementation and usage, namely in relation to tools and workflows. That is, it affects how the framework is implemented, deployed and used.

3.1 Motivation

While managing a network infrastructure, a general problem we face is to monitor device metrics so we can evaluate its health, as well as to detect possible anomalies in its behaviour. It also means that the usefulness of network packet monitoring is relevant and should be evaluated in order to enable detection of possible attacks.

As the current protocols in place at ISCTE – Instituto Universitário de Lisboa (Iscte) regarding the management of anomalies and attacks to servers are far from optimal, there was a need to study frameworks capable of filling in this gap. Indeed, this research was motivated by the need to evaluate the importance of monitoring and detecting possible anomalies and attacks on servers using open-source software, and taking advantage of *Big Data* and *Security Information and Event Management (SIEM)* methodology to obtain

results in real-time.

3.2 Proposed Architecture

The proposed framework holds two subjects of analysis: the network, where data is analysed to detect possible attacks, and the system metrics, that allows the analysis of possible anomalous behaviours, so aiming to indicate whether in a particular server something is wrong or not.

Our work extends research work carried out by Dias et al. [37], but notice that, instead of analysing only network data, we will deal with system data analysis as well.

Also, we consider that it is an advantage to favor as much as possible the concept of *Plug-in-and-Play*. That is, to build a framework around the idea that a set of modules individually implemented can work together but they can easily be replaced later on if that seems to be appropriated. Hence, we achieve capacity and flexibility in accordance to the deployment needs.

On the basis of previous considerations, the proposed architecture is mostly supported by four distinct modules, as depicted in Figure 3.1.

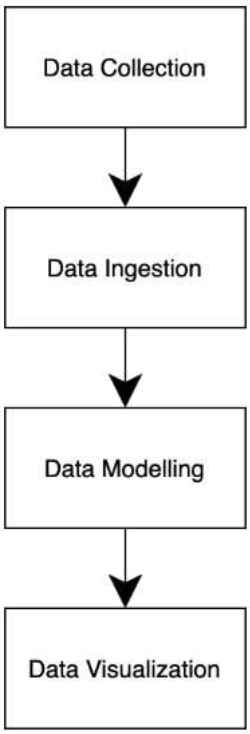

Figure 3.1: Proposed modular architecture with an underlying *Plug-in-and-Play* concept and its usability regarding data to resemble a dataflow pipeline.

First, there is the *"Data Collection"* module, where data is collected from certain data source endpoints. Then we move to the *"Data Ingestion"* module, where data is ingested

and selected and/or filtered. Next, data has to be evaluated. This evaluation implies data correlation and modelling, a process that takes place in the *"Data Modelling"* module. Finally, the results obtained will be analysed and visualized, a task that is carried out in the module *"Data Visualization"*.

A major aspect in this architecture is the use of tools, techniques and methods from the areas of *Big Data* and *SIEM* methodology. The use of ML algorithms for data modelling and the importance of scalability is critically important as well.

Notice that the implementation of this architecture fits perfectly into the *Plug-in-and-play* concept mentioned earlier on. Indeed, the design of the architecture was also driven by making sure that further adjustments in the future are possible without too much disruption. Hereafter, we will present these modules in more detail. As can be seen in Figure 3.2, each module is supported by particular tools and techniques, described in the following sections.

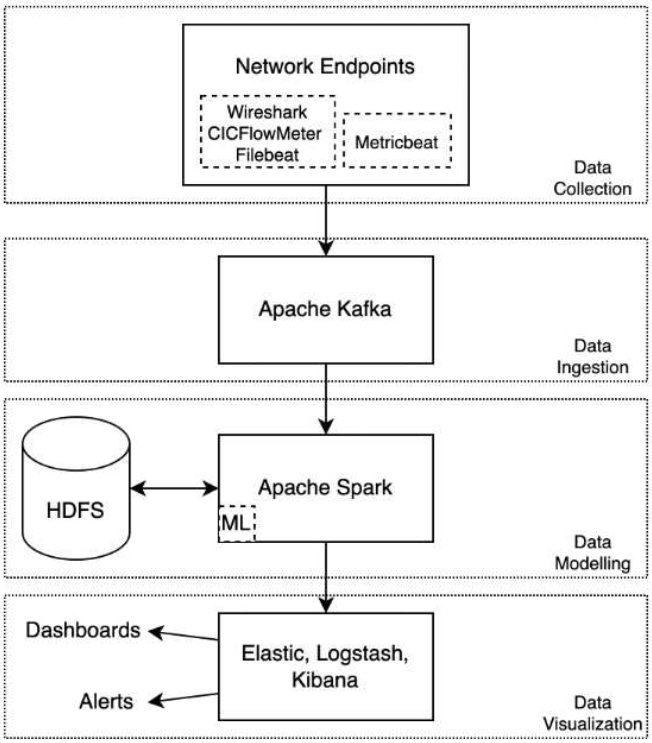

Figure 3.2: Mapping main tools and frameworks that support the proposed architecture, which highlight the *Plug-in-and-Play* concept: Each block shown in Figure 3.1 is supported by specific tools, also dependent on the type of data under consideration (network/system metrics).

3.3 Data Collection

The data collection module is about collecting data of interest that will be further down analysed. Basically there are endpoints supplying data, according to the category of data.

For network data, we elect primarily tools like Wireshark [100] and CICFlowMeter [30] to provide these data endpoints. As for system metrics data, we resort mainly on collection tools like Metricbeat [14].

3.3.1 Network Data

In order to collect network data from a device, the TShark tool [94] we are using (it is a terminal oriented version of Wireshark) captures network packets from the device's network interfaces in real-time. Once collected, the packets are analysed by the CICFlowMeter tool, which then transforms the raw network data into bidirectional flow instances with more than 83 statistical features.

The process to send data downstream to the *Data Ingestion* module implies to register a new instance of CICFlowMeter with a Filebeat agent [14]. Listing 3.1 shows an example of Filebeat output that will be sent downstream.

```
output.kafka:
  # initial brokers for reading cluster metadata
  hosts: ["10.83.30.31:9092"]

  # message topic selection + partitioning
  topic: 'cicflowmeter-events'
  partition.round_robin:
    reachable_only: false

  required_acks: 1
  compression: gzip
  max_message_bytes: 1000000
```

Listing 3.1: Filebeat.yml file – setting a Filebeat output, which will be sent downstream.

3.3.2 System Metrics Data

System metrics data is collected using the Metricbeat agent, which gathers different types of system data from a device. In the scope of different metrics that can be gathered, we may find important features like: (i) CPU data, indicating the statistics of CPU usage in the system; (ii) load statistic data, showing the amount of computational work done in the system between one, five and 15 minutes; (iii) memory data, measuring the memory used in the system; (iv) disk usage statistics; (v) system process state statistics such as idle, running, etc. The data is collected in a chosen time interval and is sent downstream to the *Data Ingestion* module. Listing 3.2 shows a Metricbeat output to be sent downstream.

```
output.kafka:
  # initial brokers for reading cluster metadata
  hosts: ["10.83.30.31:9092"]

```

```
  # message topic selection + partitioning
  topic: 'metrics-events'
  partition.round_robin:
    reachable_only: false

  required_acks: 1
  compression: gzip
  max_message_bytes: 1000000
```

Listing 3.2: Metricbeat.yml file – setting Metricbeat output, which will be sent downstream.

3.4 Data Ingestion

A critical aspect of ingestion of data provided by the data endpoints referred to in the previous section, relates to its ability to process large amounts of data without suffering significant performance degradation. That is why data ingestion must be supported by powerful tools and techniques. Hence, *Big Data* tools are suitable for that matter, as they are very effective in handling large volumes of data. For the purpose, we have chosen to rely on Apache Kafka [55], regardless of the category of data under consideration, that is, suitable to both network and system metrics data. Notice that ultimately Apache Kafka is a messaging system suitable for real-time data, centered around the idea of publishing/subscribing topics, as mention on the related work in Section 2.2.

In this module it is also very important to evaluate not only the quantity but the quality of data. Therefore, an analysis must be done to identify which data is relevant and what added value they bring to the analysis. That is why tools or techniques capable of measuring the usefulness of the data for analysis purposes are required in this context.

In our framework, data is ingested via an Apache Kafka instance, in real-time. It serves as a channel between producers and consumers of data but somehow Apache Kafka will behave like a buffer between them. Recall Apache Kafka can deal with several topics, which generally are aiming to segregate different types of messages. In our case, network data coming from the CICFlowMeter is saved as a certain topic (*"cicflowmeter-events"*) and metric systems data coming from Metricbeat is saved as another topic (*"metrics-events"*). Furthermore, it is possible to set retention time to safeguard data in case of possible failure in the data consumers, as exemplified in Listing 3.3.

```
# Log Retention Policy

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=48

# The maximum size of a log segment file. When this size is reached a new log
    segment will be created.
log.segment.bytes=1073741824

```

```
# The interval at which log segments are checked to see if they can be deleted
    according to the retention policies
log.retention.check.interval.ms=300000
```

Listing 3.3: Example of a file setting certain server's retention time properties.

The two topics mentioned above, for network data and system metrics data respectively, are consumed by Apache Spark where the data is pre-processed. In this phase, it is performed a removal of possible invalid options (Non applicable values) and evaluated the relevant features of the message to be considered to use on *Data Modelling*. Then the data is transformed into a dataframe.

In system metrics data, Metricbeat sends messages partitioned by the different components of the device, i.e., the CPU information that occurred at time t is separated from the load information at time t. Therefore, it is necessary to combine all the data into a single dataframe.

3.5 Data Modelling

The data modelling module aims to analyse and model the data of interest that has been gathered. It relies on state-of-the-art ML algorithms [102, 3, 19] implemented and run in Apache Spark [33]. In particular, to detect anomalies (outliers) unsupervised learning algorithms are applied to the system metric data, whereas supervised learning algorithms are applied to network data in order to identify possible attacks since there exists a knowledge base of common attacks [36, 68, 57, 42]. In both cases, the results are saved in Hadoop Distributed File System (HDFS) [88] (which provides an effective storage system capable of handling with volume, speed, and variety of data) and also sent downstream for visualization purposes.

Notice that once a data model is created then it can be used later on at run-time. Next, we will discuss the algorithms that were used.

Network data. As for ML to deal with network data, and consequently to detect attacks, we have used the Random Forest algorithm [21] to classify the data, available in the Apache Spark MLlib library [31]. It follows the in-workings of the library, namely the need to assemble the data features into a vector (Apache Spark's Vector Assembler class [40]) that then goes through the classifier algorithm mentioned, yielding to values as *normal traffic* or *malicious traffic*.

System metrics data. In respect to system metrics data, we have used two clustering algorithms: *Kmeans* algorithm [90] and the Gaussian Mixture algorithm [82], both available in the Apache Spark MLlib library [32]. It follows the in-workings of the library, namely the need to assemble the data features into a vector (Apache Spark's Vector Assembler class) that then goes through the clustering algorithm mentioned, categorizing to values as *normal behaviour* or *outlier*.

3.6 Data Visualization

The derived data produced upstream, that is, the data modelling results, need to be clearly understood by users. That is the main purpose of the *Data Visualization* module.

Needless to say, a good visualization is of utmost importance for the success of the framework overall and for the understanding of what is going on in the computer network. But visualization should strike a balance between simplicity so being effective and the amount of essential information that is representing the data of concern.

Most of data visualization is created with the help of the Kibana tool [39], which allows to build different graphical representations and enabling a user-friendly experience. Figure 3.3 provides a glimpse of the dashboards that we can create on Kibana. This figure depicts the values of the different features of the data collected overtime in association with the results of the detection of outliers.

Hence, data is sent to an Elasticsearch instance [39] and then represented in the Kibana tool. In addition to real-time visual analysis, the Elastic, Logstash and Kibana (ELK) tool enables forensic searches by relying on SIEM methodology which offers log storage, analysis, and reporting for forensic needs, as well as real-time monitoring, correlation, notifications, and console views [35]. The time window can be adjusted according to the user's needs. Figure 3.4 describes the different feature values of the data collected along with the results of the two clustering algorithms at the time window selected.

3.7 Data Model Workflows

The deployment and subsequent usage of an implementation of the proposed architecture poses various challenges due to the dynamic nature of the data under consideration. One should be aware that today's data would be quite different from data to be collected in the future. Therefore, for a framework to be successful, ML algorithms and data models should be constantly improved and updated to keep up with the evolving data and the needs of users.

For example, for system metric data, a constant peak detected at the same time every day throughout the week can indicate that this event may not be an outlier but a normal device behaviour. So, the data modelling algorithm should learn that such specific data point is within the normal pattern of the system.

Another example could be that attacks are constantly evolving. For example, there will be new ways of doing stealth attacks [26]. Therefore, in that case, it is important to analyse the data, collect reports and re-train the algorithms regarding the new subtle ways of attacks.

3.7.1 Data Model Training Workflow

In general, data models are initially created and then updated over time. That is, there is an initial setup of the model, followed by continuous learning and adaptation to new

Figure 3.3: Dashboard created in Kibana. Metrics Data on *Storage* server with a time window of the last 1 hour. In this figure it is possible to observe the value of the different features of the data collected regarding the results of the detection of outliers implemented. Image from 07/04/2022.

Figure 3.4: Dashboard created in Kibana. Metrics Data on *Storage* server with a time window at 15:30h to 19:30h on 07/04/2022. In this figure it is possible to observe the value of the different features regarding the results of the detection of outliers implemented occurred during that time.

collected data, which denotes an iterative tuning process in place. Not only affecting the model itself but the algorithms as well. Indeed, updating and/or tuning algorithms as well might the relevant to achieve enhanced data models.

In the following section, we will discuss the training of both network data models and system metrics data models.

3.7.1.1 Network Data Models

Initial model. As we are aiming to figure out possible attacks, we have used the CICIDS2018 [42] database for initial training of the model. This choice is due to its diversity of assessed attacks, which is corroborate by being widely used in most studies. The database consists of seven different scenarios: *Botnet*, *Brute-Force*, *DoS*, *DDos*, *Heartbleed*, *Web attacks* and *Infiltration* of the network from inside. It comprises 50 attack machines and a victim structure with five departments, 420 machines and 30 servers. The information includes network traffic capture and extraction of 80 features from the CICFlowMeter tool.

As mentioned in Section 3.5, the classification algorithm used is the Random Forest algorithm available in Apache Spark, and following the in-workings of the library, like assembling data features into a vector. But notice that before training it is necessary to pre-process the data in order to remove possible invalid options (Non-applicable values). It also follows the standard practice of splitting data into 70% for training and 30% for testing.

The steps involved in creating the model are implemented using a *Pipeline* class of the library [66]. Its runs a sequence of multiple stages of a specific ML workflow. Each stage is either an *Estimator* or a *Transformer*. In our case, the pipeline is formed by the *Vector Assembler*, followed by the *Label Indexer*, passing through the Random Forest (RF) algorithm and finally the *Label Converter* that converts the index label – the classification – to a string label.

The pipeline with the algorithm trained is stored in HDFS for future use. Figure 3.5 shows the in/out regarding HDFS. It is also used a *MulticlassClassificationEvaluator* class [69] to evaluate the quality of the model created, which compares the labels obtained by the algorithm using the testing data with the true labels of these instances.

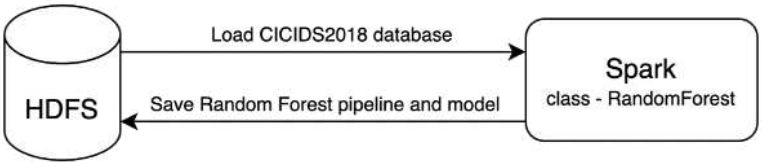

Figure 3.5: In/out of initial training model: RF classifier algorithm trained with the CICIDS2018 database to predict possible attacks.

Updating model. As the stored model is deployed and used, new data coming along can introduce unique values. It means data must be re-evaluated and eventually the model has to be updated. Hence, the Apache Spark's pipeline mentioned above that was run previously will run again. On the other hand, in this context it is also important to conduct active research on the status of cyber-attacks, therefore providing up-to-date information to allow the categorization of potential new types of attacks either attacks that are not in the initial database or new variants that emerge.

3.7.1.2 System Metrics Data Models

Initial model. Unlike the case of network data, where the publicly available CICIDS2018 database was used, there is a need to create a similar database but regarding system metrics data. Hence, we setup an initial period for data collection on each server. As each server holds a specific behaviour, the data collection has to be carried out on each device we want to detect anomalies. To do so, a Metricbeat agent was installed on the servers and each agent was sending system metrics regularly, every 10 seconds in this case. These metrics were feed to an Apache Kafka instance.

On the other side, Apache Spark was subscribing the related Apache Kafka's topic, and consuming/collecting the last 30 minutes of system metrics every 30 minutes. The collected metrics were saved by Apache Spark in HDFS. Figure 3.6 describes this collecting process.

It is based on this amount of collected system metrics data that, for example, a feature-by-feature analysis can be performed, after some pre-processing.

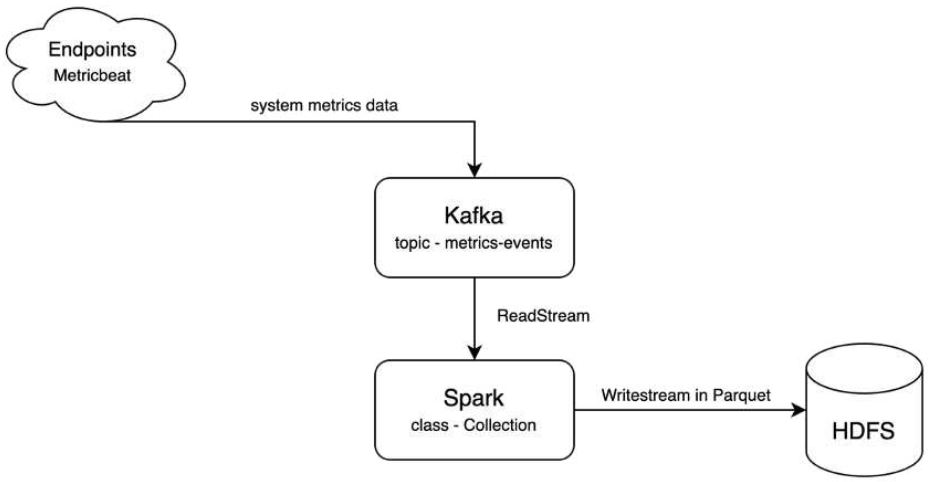

Figure 3.6: Data collection of servers metrics.

Notice that the initial gathered data from servers are separated by type of metrics. For example, a device sends a message regarding CPU metrics, a message for load statistic metrics, etc. It means that, in order to create a data model, it is necessary to previously

combine different messages that occurred in the same timestamp into a single message. Hence, to add timestamp values to anomaly detection, two new features were created: *"week"* (weekend=0, week=1) and *"labor_time"*, which considers working time from 8am to 7pm. Listing 3.4 depicts the creation of *"week"* and *"labor_time"* features.

```
result = result.withColumn("week", (when ((dayofweek(col("Date")) === 1) || (
    dayofweek(col("Date")) === 7), 0).otherwise(1)).cast(BooleanType))
            .withColumn("labor_time", (when ((hour(col("Date")) < 8) || (hour(
    col("Date")) > 19), 0).otherwise(1)).cast(BooleanType))
```

Listing 3.4: Creation of *"week"* and *"labor_time"* features.

Once the proper dataframe is created, the classic Apache Spark's data modelling process can proceed. In that respect, the algorithms used to create the system metrics data model are the K-means algorithm [90] and the Gaussian Mixture algorithm [82], both available in the Apache Spark MLlib library. The best value for the number of clusters to be used in the K-means algorithm is figured out via a silhouette function [62]. It is also calculated the cluster statistics which comprises the average distance and the standard deviation of the points in a cluster to its center.

A similar model was also created but using the Gaussian Mixture algorithm instead. Note that with the Gaussian Mixture model we expect to have a less sensitive approach to outliers since the statistical data are approximated by a mixture of Gaussians that describes the data better than distances to centroids [79, 58, 12]. Figure 3.7 points out the in/out of the unsupervised training model.

Notice that, after the models being created, all of them are stored in HDFS for further use.

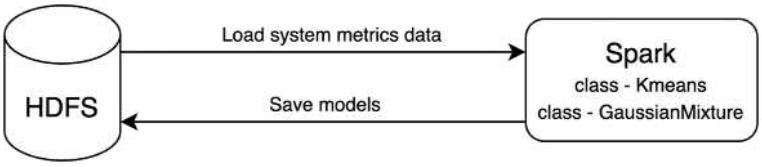

Figure 3.7: In/out of unsupervised training models for system metrics data, based on two clustering algorithms (K-means and Gaussian Mixture).

Updating model.

Similarly to the case of network data, and now because the behaviour of a machine can change over time, with the possibility of certain outliers being perceived as a pattern, it is essential to adjust the data model and eventually the underlying algorithm, according to new data.

As expected, the clustering algorithms have to be tuned to improve the categorization of machine behaviour. Again, the Apache Spark's pipeline mentioned above regarding the initial system metrics data model will run again.

3.7.2 Data Streaming Workflow

Once data models are created and then available to be deployed, it is time to run and feed theses models with data. In our case, the goal is to have the application running in real-time, in particular most concerning the detection of network attacks and anomalies.

Hence, the workflow we envisage for the purpose is to have the data being constantly streaming into an Apache Kafka instance, which acts as a buffer before Apache Spark. Then, Apache Sparks consumes the buffered data from Apache Kafka and feeds the running data models. It is up to these models to process the gathered data and provide information about the subjects at stake, say both network attacks and system metrics.

The results obtained are stored for future analysis. But, importantly, they are sent to a ELK platform to be displayed. It is with the help of the visualization tool *Kibana*, mostly via dashboards, that results are properly visually processed by users. In the end, the *Kibana* tool provides a wide range of different graphics aiming to deliver a user-friendly visualization.

4

Implementation

In this chapter, we describe the deployment procedure regarding the framework proposed in the previous chapter. First, the computational infrastructure developed is presented, where we describe the specifications of the machine provided for this purpose. It also detailed the construction of the different modules following the tools previously selected. Not least, it is outlined the data flow through the infrastructure.

Proceeding with the implementation of data modelling, the data source used for this purpose is outlined. Next, this data is subjected to the data preparation process, where we describe the transformation of the data so that it can be applied to to train the algorithms.

Finally, the implementation of the data modelling algorithms is detailed. During its implementation, we report on essential choices for the delivery of a functional algorithm. In both cases, the sequence of procedures for data modelling are very similar – ultimately it is all about following standard practices on using properly a Machine Learning (ML) pipeline.

4.1 Computational Infrastructure

We have set up a computational infrastructure for the implementation of the proposed framework described in the previous chapter 3 within Serviços de Infraestruturas Informáticas e de Comunicações (SIIC) infrastructure. It is a testbed in a server located at the the data centre of ISCTE – Instituto Universitário de Lisboa (Iscte), composed of 64 cores, 64GB of Random-access Memory (RAM) and seven Hard Disk Drive (HDD) disks, each one with 1TB. Figure 4.1 shows the infrastructure.

To test our proposed framework for attack detection, we have decided to use our own server since we did not want to interfere with the normal operation of the data centre as attacks would be performed in order to test the performance of the proposed framework. Thus, we have implemented a Virtual Machine (VM) called *"Host"*. We have deployed in this machine the free and open-source *buggy web application (bWAPP)* [22] with over 100 vulnerabilities. This app covers all major known web attacks. Notice that *bWAPP* allows security developers to conduct various application attacks.

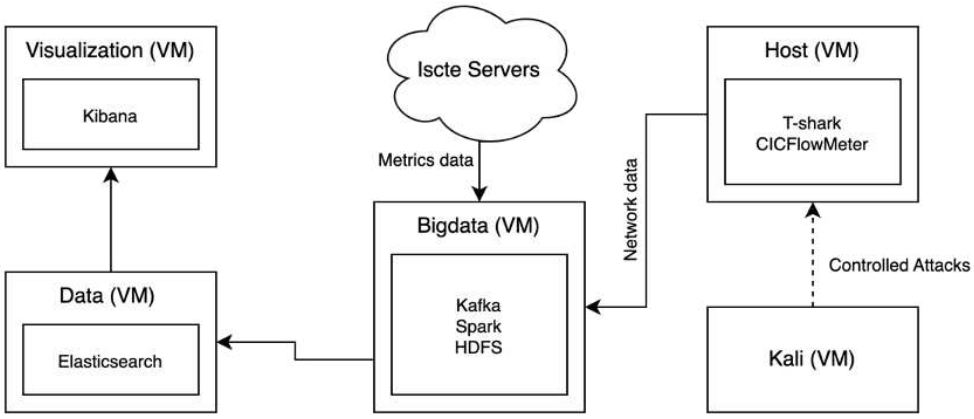

Figure 4.1: Infrastructure in the Iscte data centre for the implementation of the proposed framework.

As for deployment, we have used the *XAMPP* tool [101] to provide services like *Apache web server* and *MySQL database* that are needed to build the application. Hence we can produce network traffic data to be used for network analysis purposes.

To collect network traffic data, we create a service called "capture", as described in the bash script below (See code 4.1). This service performs a network analysis on the machine every 30 seconds using T-shark [100]. The results are then passed through the CICFlowMeter-v4 tool provided by the University New Brunswick [24], which transforms the raw network data into bidirectional flow instances.

```
#!/bin/bash
while true
do
timestamp=$(date +%Y-%m-%d_%H-%M-%S)
tshark -i ens18 -w /home/gmczo/Documents/pcap/$timestamp.pcap -a duration:30
cd /home/gmczo/Documents/CICFlowMeter-4.0/bin/
./cfm /home/gmczo/Documents/pcap/$timestamp.pcap /home/gmczo/Documents/csv/
done
```

Listing 4.1: Bash script to capture network traffic data.

To be able to generate attacks, a VM called *"Kali"* was also created. In this VM, it is used the kali operating system [53], which contains several types of useful attacks to be performed against the *"Host"* VM.

In relation to system metrics data, SIIC has provided three servers of the *Fénix* platform: *"Storage"*, *"Sql"* and *"App"*, in which a Metricbeat agent was installed to send metric data to the VM *"Big Data"*. This platform is widely used by students of Iscte for various kinds of operations: enrolling, requesting certificates, consulting their curriculum, etc. So, there is high activity in this platform, which allows us to collect data from relevant servers and to assess the importance of the usage of our framework in the academic applications.

Both network and system metrics data will be sent to the VM entitled *"Big Data"*. As its name implies, this VM is regarded as the whole Big Data processing part, where it receives the data, ingests it, models it, and stores it for batch operations.

Finally, two more VMs were created for the analysis of modelled data in real-time: *"Data"* and *"Visualization"*, both including an implementation of the architecture of the Elastic, Logstash and Kibana (ELK) tool [39]. The first machine represents the ingestion and storage of the modelled data by the *"Big Data"* VM and the second machine deploys the Kibana platform [39] that represents the data inputted in the previous machine in a user-friendly interface.

The deployment of ELK provides the possibility of assessing the results of our framework both in real-time and forensically. We have decided to decouple the tool into two machines for security concerns. Since the Kibana interface communicates with Elasticsearch [39] to display the data, it is possible to limit user activity according to their status, so not having access to certain data if that was defined. Notice that by splitting user interface from storage, we create a security barrier around the data source.

4.2 Network Data Modelling

As presented in Section 3.3.1 in the previous chapter, one of the goals is to figure out possible attacks on the network. It means the data under consideration in this case is network data. So we have to create, update, and use a data model regarding network data.

4.2.1 Source Data

First, let us consider the issue of source data for further training network data. Most of related research done worldwide uses default databases to help the process. That is why we have advocated in Section 3.3.1 the use of the popular *CICIDS2018* database, which is provided by the *University New Brunswick* [24]. This database contains 16,150,536 instances. Table 4.1 shows different labels (attacks) available in the database and the number of instances per label.

4.2.2 Data Preparation

Prior to any training of a ML algorithm, the data to be used in the task requires to make sure that it is well defined and with high quality. In this case, the process starts by removing null values. Then, a feature-by-feature analysis is performed, where we remove the timestamp. For the remaining features, temporal plots are created to analyse the data behaviour over time.

From that analysis, features with constant values or with an infinite rise are eliminated. After that, a correlation matrix is created to detect the existence of possible features with

Table 4.1: *CICIDS2018* database – number of instances per label.

Label	Number of instances
Benign	13,403,145
DDOS attack-HOIC	686,012
DDoS attacks-LOIC-HTTP	576,191
DoS attacks-Hulk	461,912
Bot	286,191
FTP-BruteForce	193,354
SSH-Bruteforce	187,589
Infilteration	161,096
DoS attacks-SlowHTTPTest	139,890
DoS attacks-GoldenEye	41,508
DoS attacks-Slowloris	10,990
DDOS attack-LOIC-UDP	1,730
Brute Force -Web	611
Brute Force -XSS	230
SQL Injection	87

a high correlation, say, 90%. In this case, using one of the these features in the algorithm is enough. The point is to have the set of features highly independent of each other.

In the end, we got 67 features to train the algorithm for attack detection in the network. The features are shown in Figures I.1 and I.2 and I.3 in Annex I).

We notice that, while reviewing the state-of-art, several times in the case of *CICIDS2018* database only DoS attacks data was analysed. Moreover, some studies presented the feature selection task.

For example, Benayas de los Santos [16] applied feature selection to prevent dimensionality issues. Based on the computed correlation matrix, the features that have shown a correlation greater than 0.5 regarding the output feature "label" were selected. The higher this value, the more impact this feature exerts on the type of attack detected.In our use case, we have decided to use all available features.

4.2.3 Data Modelling Algorithm

The problem to tackle here as far as data modelling is concerned suggests the use of a supervised learning approach. As mentioned in Section 3.5, we have decided to use the Random Forest algorithm [21] due to its easy application, higher accuracy and resilience to overfit with more features. We have set several numbers of trees while training the data, and then we chose the one showing the highest accuracy. Table 4.2 shows the results, depicting the accuracy and the training time for each number of training trees.

Table 4.2: Network data training stage – Random Forest.

ML Algorithm	Training time (s)	Accuracy
Random Forest with 10 trees	432 s	0.9496
Random Forest with 20 trees	**411 s**	**0.9568**
Random Forest with 50 trees	481 s	0.9488
Random Forest with 100 trees	591 s	0.9488

4.3 System Metrics Data Modelling

Another research goal mentioned in the previous chapter was to deal with anomalies in the system, say the servers. In that respect, in Section 3.3.2 we have stated that system metrics data will be used for that matter. Now we will be discussing in more detail the creation, updating and use of a correspondent data model.

4.3.1 Source Data

The initial data to be used to create a data model consists of system metrics data collected from the three servers in the period 22/09/2021 till 01/11/2021. But the week 25/09/2021 – 02/10/2021 was removed, as further explained in Section 5.2.1. In the end, there are 33 days of collected data from the on the three servers provided by SIIC: *"Storage"*, *"Sql"* and *"App"* server for training purposes.

4.3.2 Data Preparation

Following similar procedures as in the case of network data mentioned above, we started by removing errors from data. Indeed, sometimes during data collection, failures may occur when the information is sent. Thus, we removed null values, records without content or with unexpected content as such data is not suitable for training a model.

Then it was performed a more detailed analysis of the information contained in the data. Recall that we have CPU usage data, disk data and data of different components that are vital for a correct performance of a device.

Just to mentioning, the data is received by messages with several values, which are usually called features. Each feature has a data type and a value. For example, we have a feature called *"CPU user"*, a *double* value, that indicates the percentage of CPU used by a user inside that system, and a feature *"processes running"*, an *integer* value, that shows the number of processes running in that system. All these features must be evaluated individually to gather the final features that contribute to the analysis of the system's behaviour.

Hence, from the data collected from the Iscte servers, it was performed a feature-by-feature analysis. It starts by analysing if there are features in which the values are constant or an infinite rise. In our case, these features do not produce useful information for the creation of a clustering as they do not provide any pattern. Figure 4.2 shows a

comparison of a temporal graph of a feature that varies its value vs. a temporal graph of a feature with a constant value.

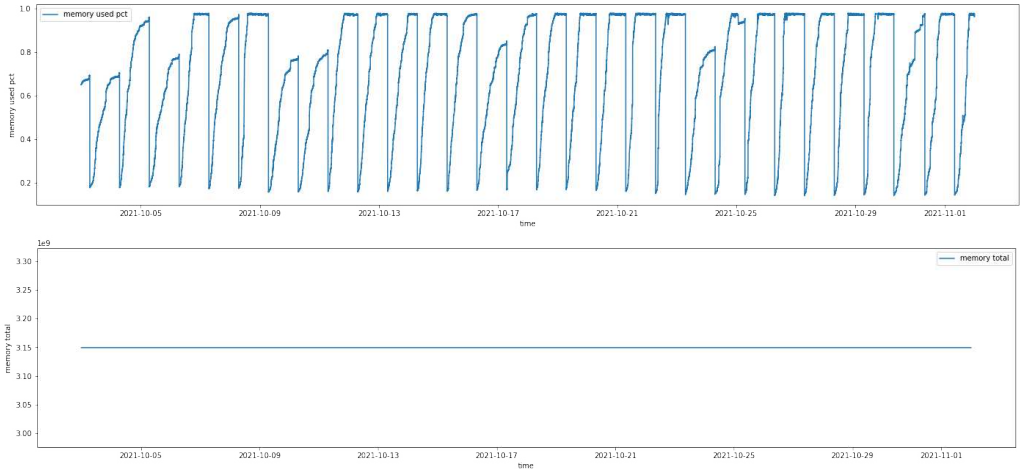

Figure 4.2: Feature with variance over time vs. feature with constant value.

Afterwards, the correlation between features was analysed. If the correlation between these features has a value greater than 90%, it indicates that these features show similar behaviour over time. Of these, only one should be considered to eliminate the repetition of information that does not benefit the modelling process (clustering).

By the end of this process, in our case study, we have obtained 31 features for the *"Sql"* and *"Storage"* servers and 57 features for the *"App"* server. The *"App"* server presents more features since the system contains three fundamental disks for evaluating the system's behaviour, whereas in the other two only one disk is considered. The features are shown in Figures I.4, I.5 and I.6 in the Annex I.

Once the features were chosen, we proceed to remove outliers from data. This step was done in order to reduce the possibility of considering points that were not the usual behaviour of the system. Notice that during the initial data collection, there was no previous study of what would be considered as normal values in the system's behaviour. Therefore, if it was a dataset in which we had more information about the system behaviour, we should take into account that certain peaks can also be normal functioning patterns.

This process of outlier removal was supported by a *z-score* function [104], which is commonly used for this type of task. This function evaluates the number of standard deviations that a given value is from the mean value in that dataset.

We have considered a value as an outlier when its z-score is greater than 3σ from the mean. The value 3σ is commonly used in the literature, as it is the most stable value to be considered for the limit of a set of values belonging to the same group [27].

Table 4.3: Numbers of instances in the training data before and after applying peak removal.

	Initial data (#)	Data after peak removal (#)
Storage server	280,338	**237,850**
Sql server	280,352	**233,068**
App server	280,190	**249,802**

4.3.3 Data Modelling Algorithms

As mentioned in Section 3.5, we have elected two clustering algorithms to create a system metrics data model: *K-means* [90] and *Gaussian Mixture* [82]. We will compare their performance and choose the best one. However, since we are dealing with anomaly detection, we might go with both of them in order to not miss an event.

K-means Algorithm

The *K-means* algorithm is a classic clustering algorithm. Prior to its application, there are two main issues to consider. Firstly, the number of clusters to be used, as it has to be set in advance. Secondly, related to performance issues, whether or not dimensional reduction techniques should be applied. Notice that, in general, if a dataset contains a very large number of features, then the performance of a ML algorithm may be drastically affected.

Number of clusters. It was applied the silhouette method [62] in the context of the *K-means* algorithm to get an appropriate number. Basically, this method evaluates the similarity of a point to its cluster and the disparity to the other clusters. This value, the closer to 1, the better the quality of the division of the points in space by the different clusters. Table 4.4 shows the silhouette value for each number of clusters regarding the three servers.

Looking at the results obtained, we can conclude that all three servers present a better silhouette if considering two clusters.

Table 4.4: Silhouette evaluation per number of clusters using *K-means*.

Clusters	Sql server	Storage server	App server
2 clusters	**0.8802**	**0.9747**	**0.9234**
3 clusters	0.8167	0.9620	0.6939
4 clusters	0.7146	0.9515	0.6330

Dimensionality Reduction. In the context of creating a model using the *K-means* algorithm, we have used the Latent Dirichlet Allocation (LDA) model [20] available in Apache Spark [32] to perform dimensional reduction in the training data of each server. The outcome was a reduction to just 10 features (default value in the Spark code). Table 4.5 shows the comparison of silhouette value in both cases, with and without the application of dimensional reduction. We can conclude that the clustering algorithm achieves

a worse silhouette when dimensional reduction is applied. Therefore, the application of LDA should be discarded.

Table 4.5: Silhouette evaluation after dimensional reduction with LDA, or not.

ML Algorithm	Sql server	Storage server	App server
K-means 2 clusters with LDA	0.1184	0.4801	0.8931
K-means 2 clusters	**0.8802**	**0.9747**	**0.9234**

Gaussian Mixture Algorithm

We have also implemented the *Gaussian Mixture* clustering algorithm [82]. As mentioned above, we were keen on comparing results from different unsupervised algorithms in order to select the best one.

For this model, we set two gaussians to give more freedom of adaptation to see if the results would improve over the *K-means* outcome, with the same number of classes.

In Table 4.6 we can be observed that the *Gaussian Mixture* algorithm can fit the data more effectively. While *K-means* can distinguish two clusters with different sets of points, *Gaussian Mixture* finds a higher difficulty in separating the points by two gaussians. This happens since a gaussian can better aggregate the points scattered through space while *K-means* has less flexibility and is more sensitive to data scattering.

Table 4.6: Comparison between models – points distribution per clusters/gaussians.

ML Algorithm		Sql server	Storage server	App server
K-means: clusters				
	0	23,723	232,329	104,270
	1	209,345	5,521	145,532
Gaussian Mixture: gaussians				
	0	233,068	32,913	249,802
	1	0	204,937	0

5

Evaluation

In this chapter, we lay out some experiments and tests we have carried out to evaluate the proposed solution. As mentioned in previous chapters, we have two categories of data to be processed and analysed – network data and system metrics data. Each one of them relates to a particular type of experimental interest or use case. That is, respectively, detection of anomalies in the system and of attacks on the network.

All the experiments regarding the two types mentioned were done using the computational infrastructure presented in the previous chapter, Section 4.1.

5.1 Network Attacks

As for detection of attacks on the network, we single out two experiments we have made: one, via launching *"slow HTTP test"* attacks [50], and a second one related to *"SQL injection"* attacks. Next, both experiments are discussed.

But first, notice that in order to access the analysis of attack detection, it is important to do it in a controlled environment, so it does not affect the institution's servers. Hence, we have used the *"Kali"* machine, which already contains several tools for networking purposes, and the attacks were created targeting the web app implemented in the *"Host"* machine. The data generated was evaluated using the implemented *Random Forest* classifier.

5.1.1 *"Slow HTTP test"* Attack

A *"slow HTTP test"* attack was carried out on the *"Kali"* machine terminal using the highly configurable tool *"SlowHTTPTest"* [91] having as victim the web app. This attack consists of sending several HTTP requests but slowly. The server waits until all data arrives, which causes a congestion in the server. As result, the web app becomes impossible to use. This attack is very simple to perform, and a poorly configured server will eventually be affected.

When this type of attack is launched, a *"DDoS-HOIC"* and *"SSH Brute Force"* attacks were detected in the *"Host"* machine. Figure 5.1 depicts the results of the attack detection

when the *"slow HTTP test"* was performed. The *"DDoS-HOIC"* attack is an open-source Denial-of-Service (DoS) attack that consists in flooding a web service with a large number of *"GET"* and *"POST"* HTTP requests. As a consequence, the server becomes extremely congested and ends up not being capable of responding to new client requests [76]. It is also detected the presence of an attack *"SSH Brute Force"*. This attack occurs when there is a brute force attempt to authenticate in remote servers via *SSH* [49, 43], which is one of the most common protocols for *IT* infrastructures.

After evaluating the attack performed, it should be noted that both consequential attacks fit into the same group of DoS attacks. However, it was not reached a distinction between them.

This outcome can be justified by the fact that, although the attack was considered *"slow HTTP test"*, it was executed in a burst, which causes immediate unavailability of the web service. This sudden blocking of the service may end up being considered another type of attack, but it is still considered a DoS attack. In other words, there is a proximity between the type of attack triggered and the attack detected by the algorithm.

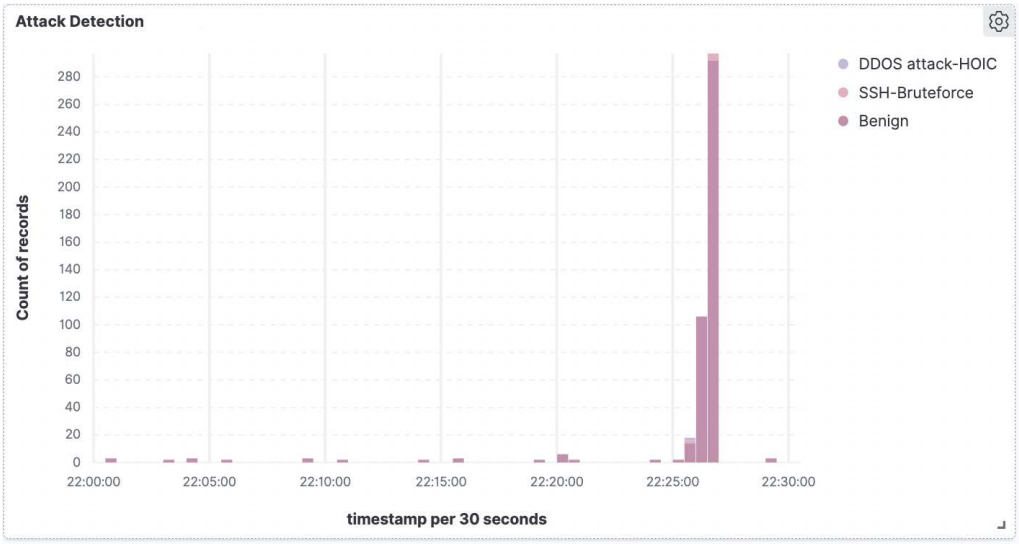

Figure 5.1: Result of network attack detection after launching a *"Slow HTTP test"* attack.

5.1.2 *"SQL injections"* Attack

A *"SQL injections"* attack consists of the exploitation of vulnerabilities in database queries, with the purpose of discovering secret information stored in the servers [97].

In this second experiment regarding network attacks, we have launched several query-based *"SQL injections"* into the application in order to access sensitive information stored in the database [23]. As consequence, occasional *"Bot"* attacks were starting to be detected. Figure 5.2 shows the results of the attack detection when a *"Sql injection"* is performed.

This *"Bot"* attack consists of small software that automates web requests for various purposes without a human intervention, like scanning the content of a website. Notice

that *"Bot"* attacks started as simple spam operations but have grown into complex attacks on infrastructures of various companies [98].

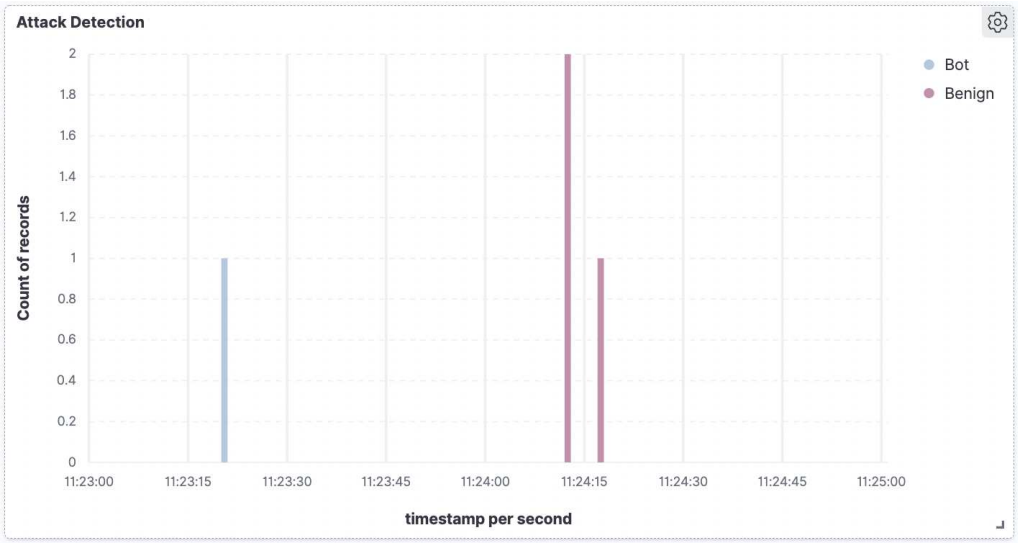

Figure 5.2: Result of network attack detection after launching a *"SQL injection"* attack.

This type of attack can be summarized as any type of operation performed on a website, i.e., the bot attack itself can contain a *"SQL injection"* attack inside. With the attacks carried out, it is possible to see that the algorithm detects that an attack is occurring in the network. However, this detection is not yet detecting the presence of the type of attack launched in the network.

In brief, the attack detection is not yet optimally identifying the type of attacks launched in the network. That should be taken into account so further research is needed in that regard. For example, it is worth considering the introduction of new data and new algorithms to achieve a more detailed analysis.

5.2 System Anomalies

In the context of detecting anomalies in the system, we have carried out several experiments, and considering different timespans as far as collected data is concerned. Then we have used these models to validate their quality and, at same time, to get insight into the behaviour of the servers under consideration, during that period.

5.2.1 Models Creation

In total, there is an initial set of 12 models that were created, four for each server (*"Storage"*, *"Sql"* and *"App"*). For each server, we have two clustering algorithms available – *K-means* and *Gaussian mixture* – and two datasets, with different time-spans, a shorter one and a normal one. Let us call them datasets A and B respectively. The timespan for dataset A is the period from 22/09/2021 to 01/11/2021, but excluding one week so there

are 33 days in total. As for dataset *B*, the period is from 22/09/2021 to 03/04/2022, again excluding one week, so there are 186 days in total.

The excluded week in both cases corresponds to the *Enrolment Week*, which runs from 25/09/2021 to 02/10/2021, and that is when all new students access the *Fénix* platform at the same time. Such usage pattern can be considered as non-normal. That is why we have excluded it. However, the data from that week still will be used for the purpose of checking the quality of the models created. That is, the created models will be applied specifically for the *Enrolment Week*.

Also, notice that by using a larger dataset we can check the impact of having more data in the models we may create. Figure 5.3 illustrates the timespan for data collection.

The process of collecting system metrics data have followed the protocols set in Sections 4.1 and 4.3.1.

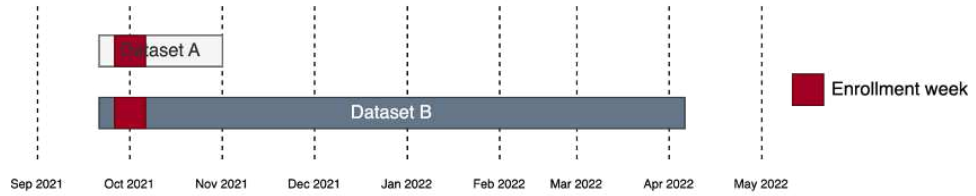

Figure 5.3: Period of time of each dataset according to the timeline used to collect system metrics data.

An important aspect relates to the need of setting the threshold at which an anomaly is considered to be the case. In the end, it is a decision function set in the clustering models.

In our implementation, the models profile the new data. The *Gaussian mixture* model calculates the probability of it belonging to a gaussian and assigns it to the one with the highest score. The *K-means* model assigns it to the closest cluster. Afterwards, we use a traditional approach of deeming an outlier when the distance to the centroid is greater than three standard deviations *(3σ)* [27]. More specifically, in the *K-means* model, this value is calculated in relation to the cluster centres, whereas for the *Gaussian mixture* model, this value is estimated using the mean point of each gaussian.

5.2.2 Enrolment Week Use Case

Giving the models created, we will now be looking at the usage pattern in the servers concerning specifically the *Enrollment Week*.

5.2.2.1 Models Based on Dataset A

When using the *Enrollment Week* in the models that were created relying on dataset *A*, we realise that, regarding time for categorization (see Table 5.1), the two models, *K-means* based and *Gaussian Mixture* based, have very efficient values. The difference is minimal.

Table 5.1: System anomalies detection on the *Enrollment Week* (seconds). Models are based on dataset *A*.

Machine Learning (ML) model	Sql server	Storage server	App server
K-means	0.21	0.39	0.27
Gaussian Mixture	0.18	0.18	0.35

Table 5.2: System anomalies detection – number of outliers. Models are based on dataset *A*.

ML model	Sql server	Storage server	App server
K-means	3,188	1,350	1,545
Gaussian Mixture	2,986	737	1,190

Also, the *K-means* model detects more outliers than the *Gaussian Mixture* model. As expected, this happens due to its higher sensitivity to outliers and the fact that a gaussian can have more scattered data across the space.

We have built a dashboard in the *Kibana* tool to visualise the collected metrics data in each server, as well as the results delivered by the anomaly detection models. Two bar charts were created in this dashboard. Each one of then shows the number of outliers detected in each model. We can set the window time we want since we're on a Security Information and Event Management (SIEM) based platform. By default, the platform shows the data of the last 15 minutes.

With the help of these charts, we can figure out at which moments abnormal values were detected, and if there is any relationship between the results of the two models. Figure 5.4 represents the detection of outliers in the *"App"* server between 25/09/2021 at 00:00h to 27/09/2021 at 23:30h. From there, we conclude that the detection depends a lot on the model used. The *K-means* model shows higher sensitivity, i.e. higher number of outliers, when compared with the *Gaussian Mixture* model.

Since the new students' enrolment began on the weekend of 25/09/2021 and 26/09/2021, it is expected that there will be higher affluence of traffic from this date on, which will cause higher metrics values comparing to a normal day, so producing an abnormal behaviour in the servers. Table 5.3 shows the hour of the day that the largest outlier peak was registered in the *Enrollment Week*.

From this table, in most of the days, both models have detected the largest peaks at the same time slots. This synchrony gives us an idea of reliability in the categorization of abnormal behaviour compared to the normal operation of a device.

It is noticeable that the peaks detected in the *"Sql"* server occur consecutively at the same hour of the day (05:00h). This indicates that these outliers have a pattern, which means they may not be an anomaly but rather it is representing a normal system behaviour, probably related to processes being executed at that time that are overloading the server.

To explore in further detail the effectiveness of clustering models, we have decided

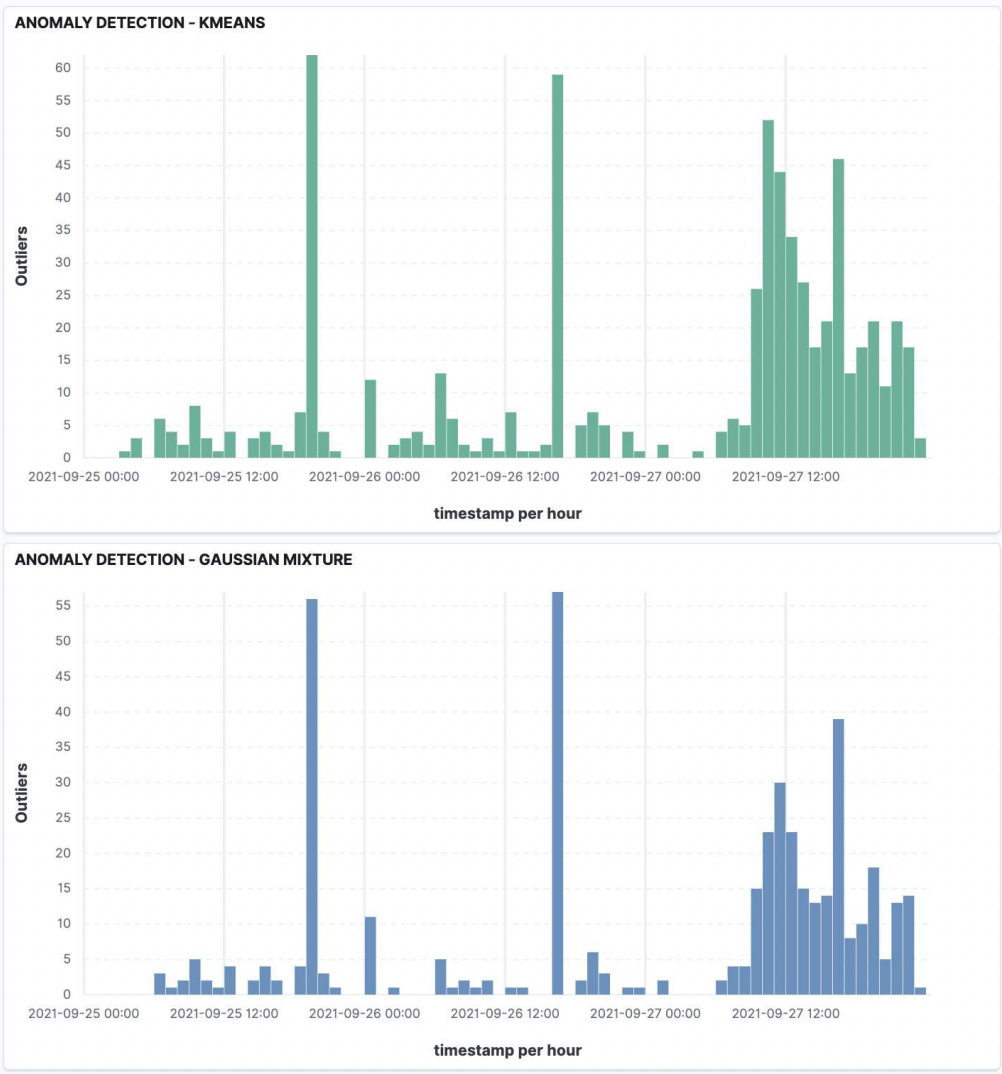

Figure 5.4: Anomaly detection in the *"App"* server, with a time window from 25/09/2021 at 00:00h to 27/09/2021 at 23:30h.

to conduct a more in-depth analysis of what might have triggered a specific outlier peak. Among the various peaks depicted in Table 5.3, we have selected the peak that occurred on 27/09/2021 at 10:00h in the server registered, for the *K-means* model.

Based on the outliers detected in this peak, and the training data of the model, it is possible to verify that the feature *"memory actual used"* presents values that are not part of the pattern of any cluster. We can conclude that, in this hour, there was unusual behaviour due to the abnormal value of memory normally used in that server (see Figure 5.5).

These peaks show that the model is successfully categorizing anomalous behaviour during the week of system overload. This in-depth observation allows us to verify that it is possible, with further and detailed analysis, to understand the results obtained by anomaly detection we have implemented.

Table 5.3: Hour of the biggest peak of outliers detected on a given day. Time window of the last 24 hours of each day.

Model	Sql server	Storage server	App server
K-means			
25-Sep-2021	05:00	09:00	19:00
26-Sep-2021	05:00	22:00	16:00
27-Sep-2021	05:00	10:00	10:00
28-Sep-2021	05:00	16:00	17:00
29-Sep-2021	05:00	15:00	11:00
30-Sep-2021	05:00	10:00	08:00
01-Oct-2021	05:00	14:00	11:00
02-Oct-2021	05:00	12:00	17:00
Gaussian Mixture			
25-Sep-2021	05:00	21:00	19:00
26-Sep-2021	05:00	12:00	16:00
27-Sep-2021	05:00	15:00	16:00
28-Sep-2021	05:00	14:00	17:00
29-Sep-2021	05:00	15:00	11:00
30-Sep-2021	05:00	10:00	08:00
01-Oct-2021	05:00	14:00	11:00
02-Oct-2021	05:00	12:00	17:00

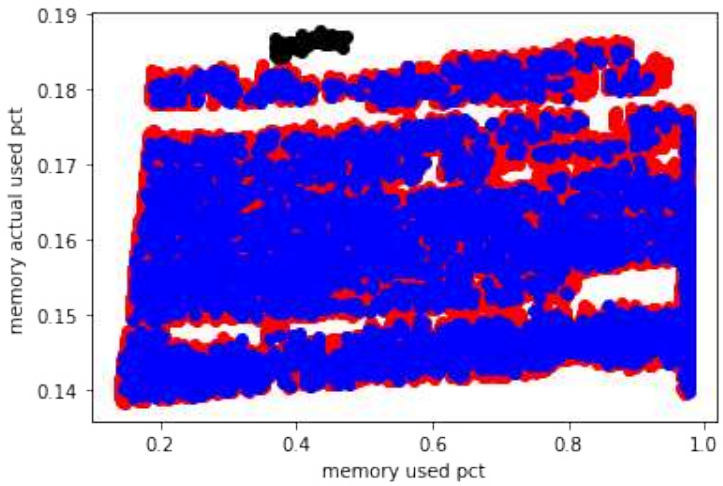

Figure 5.5: Memory actual used vs. memory used in the *"Storage"* server. The red/blue points represent the training points of each cluster whereas the black points represent the outliers registered on 27/09/2021 at 10:00h.

Table 5.4: Number of outliers detected on *Enrollment Week* on *"Storage"* server.

	Storage server	
	K-means	Gaussian Mixture
Dataset A	3,188	2,986
Dataset B	3,031	2,749
Model Tuning	1	14

Table 5.5: Number of outliers detected on *Enrollment Week* on *"Sql"* server.

	Sql server	
	K-means	Gaussian Mixture
Dataset A	1,350	737
Dataset B	1,171	459
Model Tuning	0	0

Table 5.6: Number of outliers detected on *Enrollment Week* on *"App"* server.

	App server	
	K-means	Gaussian Mixture
Dataset A	1,545	1,190
Dataset B	668	672
Model Tuning	29	27

Both models detect the same main events, which corroborates that the proposed solution can detect outliers. We notice however that the observed results are influenced by the threshold set (3σ), which affects the detection of outliers.

The improvement of the models should be a continuous process parallel to real-time anomaly detection. Therefore, it is important to carry out further study regarding the threshold value and to retrain the learning algorithms according to the new data being collected over time. Thus, for example, with the continuous tuning of the ML models.

5.2.2.2 Models Based on Dataset B

We are now repeating the analysis regarding the *Enrolment Week*, but using the models that were created based on the normal dataset *B*. That is, models based on more data in comparison to *A*. We want to figure out in particular if there is substantial influence of the size of data on the clustering models.

When comparing both situations, in the dataset B situation there is in general a decrease in outliers. This is an expected result since the new models are re-trained with more data, which is assumed to tune the categorization.

Tables 5.4, 5.5 and 5.6 show the number of outliers detected for each case. In general, there was not a significant change, which is positive in a certain way, as it indicates that our models were properly modelled in the first experiment (with dataset *A*).

5.2.3 Model Tuning Experiment

Despite the reduction of outliers regarding the *Enrollment Week* in the situation mentioned in Section 5.2.2.2, we still think the number of outliers is quite substantial for what we would expect. Thus, we have decided to further investigate on the possible reason as to why there is such high number of outliers, and if that number was supposed to be the case.

Thereafter, we proceeded to evaluating the models as a new use case, picking up a random week of data but considered to be a normal one, as far as usage pattern is concerned. It turns out we have used data from 12/03/2022 to 19/03/2022. The number of outliers detected for this week is still considerable, which is not desirable for detection as it looks like the models are very immature. These results can be justified by the removal of relevant data when applying the *z-score* function in the training data, which contains a fixed threshold of 3.

This value in the context of considerably small training data could remove important data that would be a normal usage pattern. But, since relative to the majority of the data, they are small amounts, they end up being removed with the *z-score* function. Therefore, the *z-score* value should be re-evaluated in future research. To validate this assumption, we have created new models using the dataset *B* from the experiment 5.2.2.2 but now removing the *z-score* method from the ML pipeline.

After creating these models, we have used them in the *Enrollment Week*, and similarly to what we have done in the previous experiments. We have concluded that the number of outliers was reduced very significantly. This outcome highlights the importance of the correct application of this technique, which will directly affect the categorization of the models, both in terms of false positives and false negatives.

This new experiment also serves as verification that the constant spikes detected on the *"Sql"* server were a usual pattern of the server and not outliers as we speculated.

Table 5.7 demonstrates that both the *K-means* model and *Gaussian Mixture* model do not detect outliers on the *"Sql"* server.

Furthermore, it is noted that the peaks of outliers regarding the experiment 5.2.2.2 changed to a much lower number of occurrences, which is expected given the wider coverage of the models. By analysing the results, we notice that the behaviour of the *"Storage"* server is more unpredictable than the behaviour of the *"App"* server, which presents greater conformity between the peaks detected by the different models. This can be checked in Table 5.7.

However, we realise that this experiment is only an attempt to validate some theories that were introduced throughout the study of the models. In a real case scenario, the *z-score* function should not be abruptly removed, since the training data is not being formalized and validated, and it may contain outliers and these outliers should not be included in the training dataset. Therefore, the most appropriate thing to do is a more meticulous study, such as calculating the best *z-score* for each feature, and comparing the

Table 5.7: Hour of the biggest peak of outliers detected on a given day on the second iteration without *z-score* function. Time window of the last 24 hours of each day.

Model	Sql server	Storage server	App server
K-means			
25-Sep-2021	—	—	19:00
26-Sep-2021	—	—	16:00
27-Sep-2021	—	—	—
28-Sep-2021	—	16:00	08:00
29-Sep-2021	—	—	06:00
30-Sep-2021	—	—	08:00
01-Oct-2021	—	—	—
02-Oct-2021	—	—	06:00
Gaussian Mixture			
25-Sep-2021	—	09:00	19:00
26-Sep-2021	—	12:00	16:00
27-Sep-2021	—	—	—
28-Sep-2021	—	16:00	08:00
29-Sep-2021	—	14:00	—
30-Sep-2021	—	11:00	08:00
01-Oct-2021	—	11:00	11:00
02-Oct-2021	—	—	—

results of the different training datasets in outlier detection.

Beyond these concerns, and since there is no previously studied database, more data has to be collected for model creation. The larger the amount of data, the wider the distribution of different types of data, and the better the ability of functions like *z-score* not removing certain values that may be patterns rather than outliers. That said, the whole process of creating models is a process that should be done wisely.

5.3 Data Visualization

Besides data visualization specifically related to network attacks and system anomalies detection, we have also created a range of charts with different features collected. Figure 5.6 shows an example of the kind of visualization obtained, comprised of graphics of cpu percentages, graphics of load values, graphics of memory and disk usage, and graphics of the state of processes on the server. Also Figure 5.7 represents an example of the visualization applied on network data, containing graphics of the source and destination Internet Protocol (IP) of the network data collected on the device.

This type of dashboard allows the user not only to study the behaviour of the models, but also to observe the evolution of the features in real-time, in a simply and pragmatically way. Indeed, with the help of these graphics it is more perceptible to visualise the variation of the features and how they may affect the results.

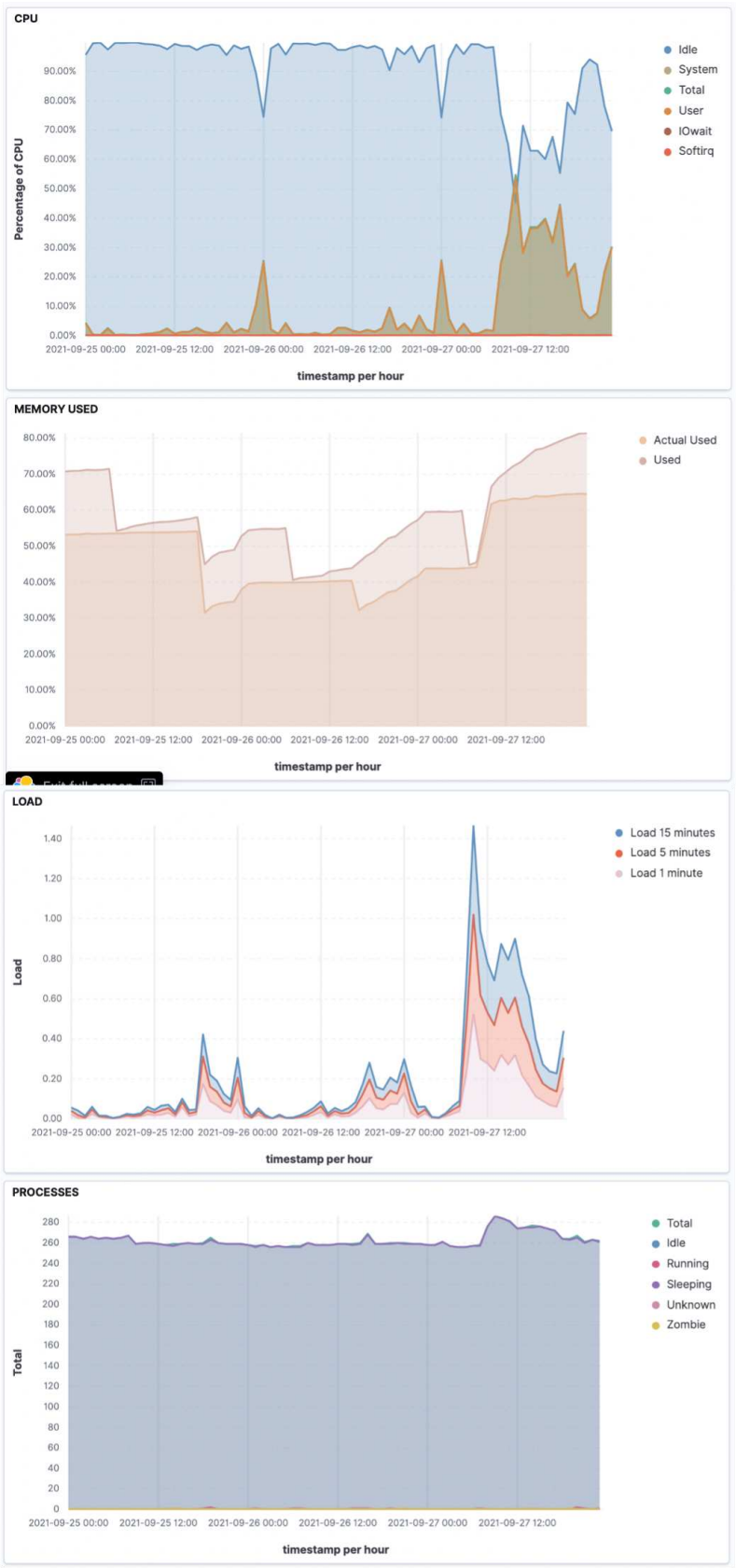

Figure 5.6: Example of a dashboard in Kibana – metrics data with a time window of 25/09/2021 at 00:00h to 27/09/2021 at 12:00h.

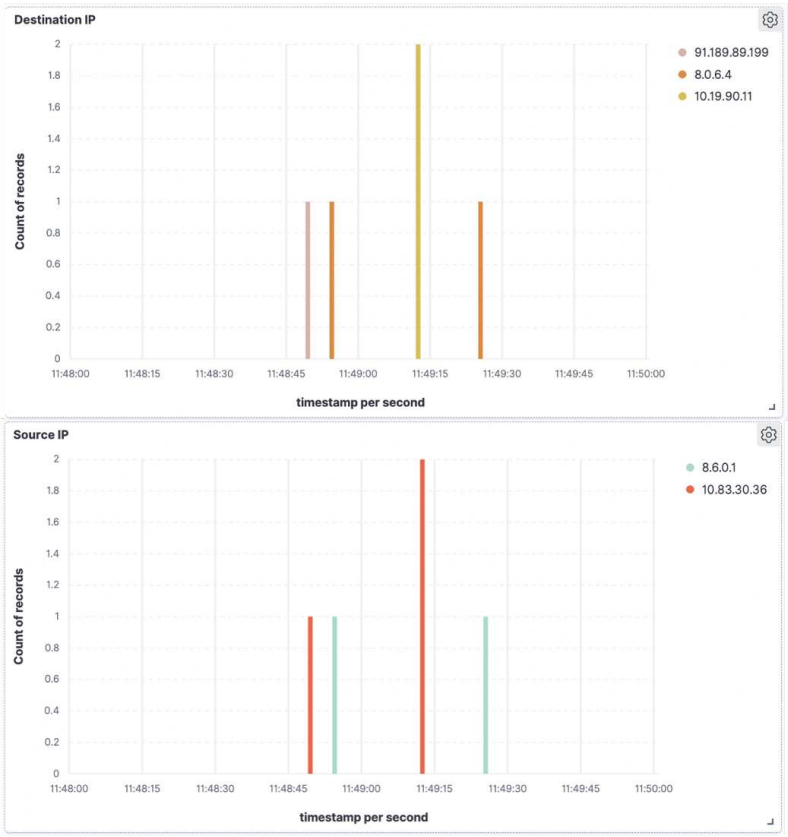

Figure 5.7: Example of a dashboard in Kibana – network data with a time window of the last 2 minutes, in real-time.

5.4 Summary

In this chapter, we evaluate the implementation of the proposed framework. This work focused on building a data streaming framework, that is, a network analysis framework working in real-time, based on the *Plug-in-and-Play* concept, where each module can be separated and adapted to the case study.

Throughout this chapter, it is possible to see that the results we have, apparently may not be surprising, but they are relevant and indicate that this approach has a future. All the models created achieved results within expectations, however, the desired result was not always delivered.

On the attack detection dimension, this evaluation serves as a proof of concept since the attacks are typified in the dataset. In this initial phase of construction of the proposed framework, the idea is also to show that these classic methods work in this architecture. There were difficulties in classifying the appropriate attack. Nevertheless, it is assigned an attack with properties similar to the real attack.

Regarding the outlier detection dimension, when analysing the data, there is a sameness in the models created regarding the two data sets used. Despite this, it is noticeable that model tuning is a very relevant task to provide the user with a more reliable system.

Besides the importance of studying attacks on the network, it is also essential to develop techniques capable of monitoring the behaviour of devices to supplement the attack detection work. This dimension is also relevant since it is also vital to monitor metrics data to maintain the health of the device to provide a good user experience.

6

Conclusions and Future Work

6.1 Conclusions

With the continuous growth of traffic over networks, attacks and anomalies incidents have been rise in several organisations. Therefore, it is important to bear in mind that an organisation must have mechanisms capable of providing cybersecurity in its network in real-time.

In this scope, we proposed a *Plug-in-and-play* framework based on data streaming. The framework was built around this concept so that all modules were split up so that the tools used in them could be changed according to the use case. During the development, we considered that it was important to create a system capable of corresponding to the current data streaming growth. Therefore, it was relevant to apply *Big data* and *Security Information and Event Management (SIEM)* methodology in our approach.

In addition to the different techniques applied for the development of the framework, we introduce two data sources that can work together, improving the security assessment of the network under study. In our proposal, we divided the topic into two dimensions, an attack dimension and an anomaly dimension in order to obtain a product that considered not only typical attacks but also the good functioning of the system analysed, an important factor in cybersecurity evaluation.

Moreover, it is always taken into consideration the classical CRISP-DM [28] process that remits us to do a constant process of evaluation of our created models, operating in parallel with the actual deployment, to deliver an up-to-date network analysis system that can evolve and obtain more reliable results.

After the implementation, we proceeded to evaluate our artefact. We performed assessment tests. Regarding attack detection, we produced controlled attacks. We analysed that our attack model detected attacks in certain experiments, however, in certain cases, we did not obtain the expected results.

Concerning anomaly detection, we decided to apply the categorization models developed in a specific week, *Enrollment Week*, a week with system data patterns considered abnormal. We observed that anomaly detection did occur. However, there were certain

data that after our analysis, we identified might not, in fact, be anomalies.

This work served as a proof of concept of the creation of a framework capable of handling data streaming and presenting real-time results in a dashboard using all the current techniques described above. This dissertation also served as a complement to the work produced by Dias et al. [37] to introduce the importance of monitoring system metrics data, since there are certain advanced attacks, stealth attacks [26], that can be carried out at such a slow cadence that they may not be detected by monitoring network data.

A goal of this research was also to answer the 5 research questions that initiated the study of the implementation of an network analysis system in the ISCTE – Instituto Universitário de Lisboa (Iscte) network. Those questions are now addressed below.

1. *What type of techniques for the development of a network analysis framework in real-time should be applied in this research?*

   As the initial knowledge on the subject was non-existent, it was necessary to research the different techniques applied in the development of a system of this nature. Throughout the research, different techniques such as system based on *SIEM* methodology, *Big Data*, *Machine Learning (ML)* and *Plug-in-and-play* were understood. The system based on *SIEM* methodology are an important and relevant typology since it is vital to have a system that not only indicates the detection of problems but also allows us to evaluate the data forensically.

   Being a project that involves a large amount of data, it is important to bear in mind *Big Data* techniques that allow the production of a scalable solution to extend the framework to as many devices as required. For the detection of attacks and anomalies, we need to apply *ML* techniques to analyse the collected data.

   The conception of this system also had the purpose of gathering different open-source tools in one ecosystem in which all could work in unison, but at the same time be independent of each other. Thus, the importance of the *Plug-in-and-play* concept emerges. With this notion of independence, it would be possible to change various parts of the union if the case study demanded it, i.e., different devices or types of data were added to the framework. Bearing this sensibility, it would be possible to build a robust system.

2. *Are there any advantages in implementing a network analysis framework for simultaneous attack and anomaly detection?*

   During the research of the related work, it was observed that many of the articles concerning the development of real-time security strategies used only data from network flows. However, when studying the topic, it was created the sensibility that in terms of attacks, increasingly there are silent attacks, stealth attacks, which consist of attacks that infect devices, but their proliferation is at a low cadence that

they go unnoticed in the network flow data analysis. Normally these attacks are hosted and use system resources for their objective.

Consequently, we thought it would be interesting to deploy a framework that could be more complete than just flow data. Throughout the research, we sought to find new data that could supplement it. Therefore, we realized that from system metrics data such as *CPU*, *disk*, *memory*, etc. it is possible to create mechanisms capable of learning the normal behaviour of a system.

From this behaviour, we could evaluate when unexpected behaviour occurs. This detection is interesting since it may indicate certain types of attacks that attack detection system do not detect. Therefore, the idea of combining attacks and anomalies detection in the same system serves to somehow add trust value to the framework.

3. *How to measure the results obtained in the proposed framework deployed on the Iscte network?*

   Based on the various articles reviewed in the related work, it is possible to verify that the evaluation of the quality of detection of possible attacks or anomalies is a challenging process. During the initial iterations of a framework, it is expected that both false positives and false negatives will be detected, ie, detected attacks or anomalies that are not and attacks or anomalies that may go unnoticed respectively. So, this should be a continuous evaluation process.

   The proposed framework keeps these aspects in mind by developing the notion of Model Tunning. This notion enables the framework to be updated and improved over time, i.e., constant improvement in the accuracy of the evaluation of the system's behaviour and constant monitoring of the emergence of new types of attacks. To perform the improvement, regarding anomaly detection, the tuning of the implemented models with the addition of new metrics data that will be collected over time is essential. In terms of monitoring new attacks, there should be constant research of articles and reports on the topic to keep up with current events.

4. *Are there benefits in the implementation of the proposed framework on the Iscte network?*

   It was seen in the related work that measuring the benefits of a network analysis framework is challenging. Particularly in the short-term as the use case of our research. Therefore, the benefits cannot be quantified, however, it is possible to qualify that this framework introduces new forms of network analysis, which until now did not exist in the organisation. However, the benefits can be assessed in the long-term with future studies in this area, evaluating the impact this framework has achieved in the Iscte network. Nevertheless, the development of this study already benefits the Iscte, highlighting the importance of the subject of security for the future.

6.2 Limitations

Concerning the results achieved in Chapter 5, we can note that there are certain limitations in our models. When we ran our experiments, in some cases, the results were not what we expected. It is noticeable that the applied models need improvement.

This is a problem that occurs naturally during the early stages of implementing ML models. But the whole structure from data collection to results presentation is operational and this aspect was already considered during the construction of our framework. It is a matter of tuning models.

However, these limitations are relevant and necessary for this approach to be of interest for future studies.

6.3 Future Work

As mentioned several times, the proposed framework is a proof-of-concept data streaming monitoring system built with open-source tools. As such it will need continuous improvement to become more reliable for users, but the important thing is that the concept worked.

Our evaluation of the proposed framework has shown that there are several improvement factors in the proposed framework, such as: (i) the adequate calculation of the *z-score* function for data cleanup in training datasets; (ii) re-evaluate the number of clusters and gaussians using adequate techniques when the models are re-trained with new data (iii) re-evaluate the deeming of an outlier.

One way to further improve the data modelling could be the application of new types of ML models. During the research, we saw several applications of deep learning. It would be an interesting path to follow and evaluate its potential in the given context against the models implemented in this dissertation.

Moreover, it would also be interesting to explore new types of devices in the network as well as the introduction of new metrics that could be relevant for the assessment of a system's behaviour to make the detection even more sophisticated. For that, it is advised the server used to implement the framework have more powerful hardware than the one used in this dissertation.

Last but not least, it can be interesting for anomaly detection and even for attacks to have both types of data combined. A third dimension that has not been explored but can be implemented in the future.

In a conclusion, it would be worthwhile to make a long-term evaluation of the benefits of this framework in the Iscte network.

Bibliography

[1] B. Abolhasanzadeh. "Nonlinear dimensionality reduction for intrusion detection using auto-encoder bottleneck features". In: *2015 7th Conference on Information and Knowledge Technology (IKT)*. 2015, pp. 1–5. DOI: 10.1109/IKT.2015.7288799 (cit. on p. 8).

[2] F. Abri et al. "Can Machine/Deep Learning Classifiers Detect Zero-Day Malware with High Accuracy?" In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 3252–3259. DOI: 10.1109/BigData47090.2019.9006514 (cit. on p. 8).

[3] M. Ahmed, A. Naser Mahmood, and J. Hu. "A survey of network anomaly detection techniques". In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2015.11.016. URL: https://www.sciencedirect.com/science/article/pii/S1084804515002891 (cit. on pp. 17, 18, 20, 30).

[4] M. Albahar. "Cyber Attacks and Terrorism: A Twenty-First Century Conundrum". In: *Science and Engineering Ethics* 25.4 (2019), pp. 993–1006 (cit. on p. 1).

[5] S. Alhaidari et al. "Network Traffic Anomaly Detection Based on Viterbi Algorithm Using SNMP MIB Data". In: *Proceedings of the 2019 3rd International Conference on Information System and Data Mining*. ICISDM 2019. Houston, TX, USA: Association for Computing Machinery, 2019, pp. 92–97. ISBN: 9781450366359. DOI: 10.1145/3325917.3325928. URL: https://doi.org/10.1145/3325917.3325928 (cit. on p. 10).

[6] S. M. Ali et al. "Big data visualization: Tools and challenges". In: *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. 2016, pp. 656–660. DOI: 10.1109/IC3I.2016.7918044 (cit. on p. 13).

[7] H. Almohannadi et al. "Cyber Threat Intelligence from Honeypot Data Using Elasticsearch". In: *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. 2018, pp. 900–906. DOI: 10.1109/AINA.2018.00132 (cit. on p. 23).

[8] M. Almseidin et al. "Evaluation of machine learning algorithms for intrusion detection system". In: *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*. 2017, pp. 000277–000282. DOI: 10.1109/SISY.2017.8080566 (cit. on p. 19).

[9] N. S. Arunraj et al. "Comparison of Supervised , Semi-supervised and Unsupervised Learning Methods in Network Intrusion Detection System (NIDS) Application". In: *Anwendungen Und Konzepte Der Wirtschaftsinformatik (AKWI)* (2017) (cit. on pp. 18, 19).

[10] M. A. B and A. N. Mahmood. "Improved Information Theoretic Co-clustering". In: *SecureComm* (2015), pp. 204–219. DOI: 10.1007/978-3-319-23802-9 (cit. on p. 18).

[11] J. B. Babu, S. Prasad, and G. S. Prasad. "Detecting and analyzing the malicious linux events using filebeat and ELK stack". In: *International Journal of Engineering and Advanced Technology* (2019). ISSN: 22498958 (cit. on p. 23).

[12] U. Baid, S. Talbar, and S. Talbar. "Comparative Study of K-means, Gaussian Mixture Model, Fuzzy C-means algorithms for Brain Tumor Segmentation". In: *Proceedings of the International Conference on Communication and Signal Processing 2016 (ICCASP 2016)*. Atlantis Press, 2016, pp. 583–588. ISBN: 978-94-6252-305-0. DOI: https://doi.org/10.2991/iccasp-16.2017.85. URL: https://doi.org/10.2991/iccasp-16.2017.85 (cit. on p. 36).

[13] E. Baseman et al. "Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs". In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2016, pp. 882–885. DOI: 10.1109/ICMLA.2016.0158 (cit. on p. 12).

[14] *Beats: Agentes de dados para o Elasticsearch | Elastic*. https://www.elastic.co/pt/beats/. Accessed: 2021-01-20 (cit. on pp. 13, 28, 77).

[15] M. C. Belavagi and B. Muniyal. "Performance Evaluation of Supervised Machine Learning Algorithms for Intrusion Detection". In: *Procedia Computer Science* 89 (2016). Twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India, pp. 117–123. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.06.016. URL: https://www.sciencedirect.com/science/article/pii/S187705091631081X (cit. on pp. 18, 19).

[16] F. Benayas de los Santos. "Design of an architecture for Cyber-Attack Detection on an SDN environment". MA thesis. 2019 (cit. on p. 42).

[17] M. P. Bharati and S. Tamane. "NIDS-Network Intrusion Detection System Based on Deep and Machine Learning Frameworks with CICIDS2018 using Cloud Computing". In: *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*. 2020, pp. 27–30. DOI: 10.1109/ICSIDEMPC49020.2020.9299584 (cit. on p. 8).

[18] S. Bhatt, P. K. Manadhata, and L. Zomlot. "The Operational Role of Security Information and Event Management Systems". In: *IEEE Security Privacy* 12.5 (2014), pp. 35–41. DOI: 10.1109/MSP.2014.103 (cit. on p. 1).

[19] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools". In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 303–336. DOI: 10.1109/SURV.2013.052213.00046 (cit. on pp. 17, 30).

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022 (cit. on p. 45).

[21] L. Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on pp. 30, 42).

[22] *bWAPP*. https://sourceforge.net/projects/bwapp/files/bWAPP/. Accessed: 2021-10-20 (cit. on p. 39).

[23] *BWAPP SQL INJECTION GET/SEARCH*. https://rioasmara.com/2021/05/02/bwapp-sql-injection-get-search/. Accessed: 2021-11-20 (cit. on p. 48).

[24] *Canadian Institute for Cybersecurity | UNB*. https://www.unb.ca/cic/. Accessed: 2021-04-17 (cit. on pp. 40, 41).

[25] A. A. Cárdenas, P. K. Manadhata, and S. P. Rajan. "Big Data Analytics for Security". In: *IEEE Security Privacy* 11.6 (2013), pp. 74–76. DOI: 10.1109/MSP.2013.138 (cit. on p. 2).

[26] L. Cazorla, C. Alcaraz, and J. Lopez. "Cyber Stealth Attacks in Critical Information Infrastructures". In: *IEEE Systems Journal* 12.2 (2018), pp. 1778–1792. DOI: 10.1109/JSYST.2015.2487684 (cit. on pp. 1, 31, 62).

[27] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882 (cit. on pp. 44, 50).

[28] P. Chapman et al. "CRISP-DM 1.0: Step-by-step data mining guide". In: 2000 (cit. on pp. 25, 61).

[29] Y.-K. Chen. "Challenges and opportunities of internet of things". In: *17th Asia and South Pacific Design Automation Conference*. 2012, pp. 383–388. DOI: 10.1109/ASPDAC.2012.6164978 (cit. on p. 2).

[30] *CICFlowMeter (formerly ISCXFlowMeter)*. https://www.unb.ca/cic/research/applications. Accessed: 2021-04-17 (cit. on pp. 13, 28, 75).

[31] *Classification and regression - Spark 3.2.0 Documentation*. https://spark.apache.org/docs/latest/ml-classification-regression. Accessed: 2021-08-10 (cit. on p. 30).

[32] *Clustering - Spark 3.2.0 Documentation*. https://spark.apache.org/docs/latest/ml-clustering. Accessed: 2021-08-27 (cit. on pp. 30, 45).

[33] J. S. Damji et al. *Learning Spark, 2nd Edition*. O'Reilly Media, Inc., 2020. ISBN: 9781492050049 (cit. on pp. 15, 30).

[34] A. Dawoud, S. Shahristani, and C. Raun. "Deep Learning for Network Anomalies Detection". In: *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*. 2018, pp. 149–153. DOI: 10.1109/iCMLDE.2018.00035 (cit. on p. 8).

[35] K.-O. Detken et al. "SIEM approach for a higher level of IT security in enterprise networks". In: *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 1. 2015, pp. 322–327. DOI: 10.1109/IDAACS.2015.7340752 (cit. on pp. 1, 31).

[36] L. Dhanabal and S. P. Shantharajah. "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms". In: *International Journal of Advanced Research in Computer and Communication Engineering* 4.6 (2015), pp. 446–452. ISSN: 2319-5940. DOI: 10.17148/IJARCCE.2015.4696 (cit. on pp. 8, 30).

[37] L. Dias et al. "OutGene: Detecting Undefined Network Attacks with Time Stretching and Genetic Zooms". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11928 LNCS.830892 (2019), pp. 199–220. ISSN: 16113349. DOI: 10.1007/978-3-030-36938-5_12 (cit. on pp. 2, 5, 23, 26, 62).

[38] M. Du et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning". In: CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1285–1298. ISBN: 9781450349468. DOI: 10.1145/3133956.3134015. URL: https://doi.org/10.1145/3133956.3134015 (cit. on p. 12).

[39] *Elastic Stack: Elasticsearch, Kibana, Beats e Logstash | Elastic*. https://www.elastic.co/pt/elastic-stack/. Accessed: 2021-01-20 (cit. on pp. 16, 31, 41).

[40] *Extracting, transforming and selecting features - Spark 3.2.0 Documentation*. https://spark.apache.org/docs/latest/ml-features. Accessed: 2021-08-10 (cit. on p. 30).

[41] F. Farahnakian and J. Heikkonen. "A deep auto-encoder based approach for intrusion detection system". In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 2018, pp. 178–183. DOI: 10.23919/ICACT.2018.8323688 (cit. on p. 8).

[42] M. Al-Fawareh and M. Al-Fayoumiy. "Detecting stealth-based attacks in large campus networks". In: *International Journal of Advanced Trends in Computer Science and Engineering* 9.4 (2020), pp. 4262–4277. ISSN: 22783091. DOI: 10.30534/ijatcse/2020/15942020 (cit. on pp. 8, 9, 30, 34).

[43] *Força bruta contra serviços SSH e FTP*. https://pt.linuxteaching.com/article/brute_force_against_ssh_and_ftp_services. Accessed: 2021-11-20 (cit. on p. 48).

[44] E. Y. Gorodov and V. V. Gubarev. "Analytical Review of Data Visualization Methods in Application to Big Data". In: *JECE* 2013 (Jan. 2013). ISSN: 2090-0147. DOI: 10.1155/2013/969458. URL: https://doi.org/10.1155/2013/969458 (cit. on p. 21).

[45] M. Harikanth and P. Rajarajeswari. "Malicious event detection using ELK stack through cyber threat intelligence". In: *International Journal of Innovative Technology and Exploring Engineering* (2019). ISSN: 22783075 (cit. on p. 23).

[46] M. AL-Hawawreh, N. Moustafa, and E. Sitnikova. "Identification of malicious activities in industrial internet of things based on deep learning models". In: *Journal of Information Security and Applications* 41 (2018), pp. 1–11. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2018.05.002. URL: https://www.sciencedirect.com/science/article/pii/S2214212617306002 (cit. on p. 8).

[47] S. He et al. "Experience Report: System Log Analysis for Anomaly Detection". In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, pp. 207–218. DOI: 10.1109/ISSRE.2016.21 (cit. on p. 12).

[48] H. Hindy et al. "Utilising Deep Learning Techniques for Effective Zero-Day Attack Detection". In: *Electronics* 9.10 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9101684. URL: https://www.mdpi.com/2079-9292/9/10/1684 (cit. on p. 8).

[49] *How to gain SSH Access to Servers by Brute-Forcing Credentials*. https://null-byte.wonderhowto.com/how-to/gain-ssh-access-servers-by-brute-forcing-credentials-0194263/. Accessed: 2021-11-20 (cit. on p. 48).

[50] *How to perform a DoS attack "Slow HTTP"with SlowHTTPTest (test your server Slowloris protection) in Kali Linux*. https://ourcodeworld.com/articles/read/949/how-to-perform-a-dos-attack-slow-http-with-slowhttptest-test-your-server-slowloris-protection-in-kali-linux. Accessed: 2021-11-16 (cit. on p. 47).

[51] B. Ingre and A. Yadav. "Performance analysis of NSL-KDD dataset using ANN". In: *2015 International Conference on Signal Processing and Communication Engineering Systems*. 2015, pp. 92–96. DOI: 10.1109/SPACES.2015.7058223 (cit. on p. 21).

[52] J. Inns. "The evolution and application of SIEM systems". In: *Network Security* 2014.5 (2014), pp. 16–17. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(14)70051-0. URL: https://www.sciencedirect.com/science/article/pii/S1353485814700510 (cit. on p. 1).

[53] *Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution*. https://www.kali.org/. Accessed: 2021-08-23 (cit. on p. 40).

[54] K. Kavanagh, T. Bussa, and J. Collins. *Magic Quadrant for Security Information and Event Management*. Gartner, June 2021 (cit. on p. 16).

[55] J. Kreps, N. Narkhede, and J. Rao. "Kafka: a Distributed Messaging System for Log Processing". In: *ACM SIGMOD Workshop on Networking Meets Databases* (2011) (cit. on pp. 14, 29).

[56] Y. N. Kunang et al. "Automatic Features Extraction Using Autoencoder in Intrusion Detection System". In: *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*. 2018, pp. 219–224. DOI: 10.1109/ICECOS.2018.8605181 (cit. on p. 8).

[57] Kurniabudi et al. "CICIDS-2017 Dataset Feature Analysis With Information Gain for Anomaly Detection". In: *IEEE Access* 8 (2020), pp. 132911–132921. DOI: 10.1109/ACCESS.2020.3009843 (cit. on pp. 8, 30).

[58] H. S. Kuyuk et al. "Application of k-means and Gaussian mixture model for classification of seismic activities in Istanbul". In: *Nonlinear Processes in Geophysics* 19.4 (2012), pp. 411–419. DOI: 10.5194/npg-19-411-2012. URL: https://npg.copernicus.org/articles/19/411/2012/ (cit. on p. 36).

[59] D. Kwon et al. "A survey of deep learning-based network anomaly detection". In: *Cluster Computing* 22.1 (2019), pp. 949–961. DOI: 10.1007/s10586-017-1117-8. URL: https://doi.org/10.1007/s10586-017-1117-8 (cit. on p. 20).

[60] H. S. Lallie et al. "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic". In: *Computers and Security* 105 (2021), p. 102248 (cit. on p. 1).

[61] P. Laskov et al. "Learning Intrusion Detection: Supervised or Unsupervised?" In: *Image Analysis and Processing – ICIAP 2005*. Ed. by F. Roli and S. Vitulano. Springer Berlin Heidelberg, 2005, pp. 50–57. ISBN: 978-3-540-31866-8 (cit. on p. 18).

[62] K. Mahendru. *How to Determine the Optimal K for K-Means?* https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb. Accessed: 2021-06-13 (cit. on pp. 36, 45).

[63] A. Manna and M. Alkasassbeh. "Detecting network anomalies using machine learning and SNMP-MIB dataset with IP group". In: *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*. 2019, pp. 1–5. DOI: 10.1109/ICTCS.2019.8923043 (cit. on p. 10).

[64] D. Mauro and K. Schmidt. *Essential SNMP: Help for System and Network Administrators*. "O'Reilly Media, Inc.", 2005 (cit. on p. 9).

[65] Y. Meidan et al. "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders". In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22. DOI: 10.1109/MPRV.2018.03367731 (cit. on p. 8).

[66] *ML Pipelines - Spark 3.2.0 Documentation*. https://spark.apache.org/docs/latest/ml-pipeline. Accessed: 2021-08-10 (cit. on p. 34).

[67] M. Monshizadeh et al. "Performance Evaluation of a Combined Anomaly Detection Platform". In: *IEEE Access* 7 (2019), pp. 100964–100978. DOI: 10.1109/ACCESS.2019.2930832 (cit. on p. 7).

[68] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942 (cit. on pp. 8, 30).

[69] *MulticlassClassificationEvaluator — PySpark 3.2.0 documentation*. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.MulticlassClassificationEvaluator. Accessed: 2021-08-10 (cit. on p. 34).

[70] S. Naseer et al. "Enhanced Network Anomaly Detection Based on Deep Neural Networks". In: *IEEE Access* 6 (2018), pp. 48231–48246. DOI: 10.1109/ACCESS.2018.2863036 (cit. on pp. 20, 21).

[71] G. Al-Naymat, A. Hambouz, and M. Al-Kasassbeh. "Evaluating the Impact of Feature Selection Methods on SNMP-MIB Interface Parameters to Accurately Detect Network Anomalies". In: *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. 2019, pp. 1–6. DOI: 10.1109/ISSPIT47144.2019.9001882 (cit. on p. 10).

[72] G. Al-Naymat, M. Al-Kasassbeh, and E. Al-Hawari. "Using machine learning methods for detecting network anomalies within SNMP-MIB dataset". In: *International Journal of Wireless and Mobile Computing* 15.1 (2018), pp. 67–76. DOI: 10.1504/IJWMC.2018.094644 (cit. on p. 10).

[73] G. Al-Naymat, M. Al-kasassbeh, and E. Al-Hawari. "Exploiting SNMP-MIB Data to Detect Network Anomalies Using Machine Learning Techniques". In: *Intelligent Systems and Applications*. Ed. by K. Arai, S. Kapoor, and R. Bhatia. Cham: Springer International Publishing, 2019, pp. 991–1004. ISBN: 978-3-030-01057-7 (cit. on p. 10).

[74] Q. Niyaz et al. "A deep learning approach for network intrusion detection system". In: *EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)* (2015). ISSN: 24116777. DOI: 10.4108/eai.3-12-2015.2262516 (cit. on p. 8).

[75] H. Nourtel, C. Cerisara, and S. Cruz-Lara. "Deep Unsupervised System Log Monitoring". In: *Product-Focused Software Process Improvement*. Ed. by X. Franch, T. Mannisto, and S. Martinez-Fernandez. Cham: Springer International Publishing, 2019, pp. 545–553. ISBN: 978-3-030-35333-9 (cit. on p. 12).

[76] *O que é o canhão de íons de órbita alta (HOIC)?* https://www.cloudflare.com/pt-br/learning/ddos/ddos-attack-tools/high-orbit-ion-cannon-hoic/. Accessed: 2021-11-20 (cit. on p. 48).

[77] E. E. Papalexakis, A. Beutel, and P. Steenkiste. "Network Anomaly Detection Using Co-clustering". In: *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2012, pp. 403–410. DOI: 10.1109/ASONAM.2012.72 (cit. on p. 18).

[78] W. Paper. "Using Splunk to Develop an Incident Response Plan". In: (2020). URL: https://www.splunk.com/en_us/form/develop-an-incident-response-plan.html (cit. on p. 16).

[79] E. Patel and D. S. Kushwaha. "Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model". In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet'19), pp. 158–167. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.017. URL: https://www.sciencedirect.com/science/article/pii/S1877050920309820 (cit. on p. 36).

[80] K. Peffers et al. "A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77. DOI: 10.2753/MIS0742-1222240302. eprint: https://doi.org/10.2753/MIS0742-1222240302. URL: https://doi.org/10.2753/MIS0742-1222240302 (cit. on p. 3).

[81] O. Podzins and A. Romanovs. "Why SIEM is Irreplaceable in a Secure IT Environment?" In: *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. 2019, pp. 1–5. DOI: 10.1109/eStream.2019.8732173 (cit. on p. 1).

[82] D. A. Reynolds. "Gaussian mixture models." In: *Encyclopedia of biometrics* 741.659-663 (2009) (cit. on pp. 30, 36, 45, 46).

[83] L. R. A. Ribeiro. "Data Analytics : Abordagem para Visualização da informação". In: (2015) (cit. on p. 3).

[84] N. Sameera and M. Shashi. "Deep transductive transfer learning framework for zero-day attack detection". In: *ICT Express* 6.4 (2020), pp. 361–367. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.03.003. URL: https://www.sciencedirect.com/science/article/pii/S2405959519303625 (cit. on p. 8).

[85] S. S. Sekharan and K. Kandasamy. "Profiling SIEM tools and correlation engines for security analytics". In: *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 2017, pp. 717–721. DOI: 10.1109/WiSPNET.2017.8299855 (cit. on p. 16).

[86] R. A. Shaikh and S. V. Shashikala. "An Autoencoder and LSTM based Intrusion Detection approach against Denial of service attacks". In: *2019 1st International Conference on Advances in Information Technology (ICAIT)*. 2019, pp. 406–410. DOI: 10.1109/ICAIT47043.2019.8987336 (cit. on p. 8).

[87] N. Shone et al. "A Deep Learning Approach to Network Intrusion Detection". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (2018), pp. 41–50. DOI: 10.1109/TETCI.2017.2772792 (cit. on p. 8).

[88] K. Shvachko et al. "The Hadoop Distributed File System". In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010), pp. 1–10 (cit. on pp. 16, 30).

[89] *Simple Network Management Protocol*. https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol/. Accessed: 2021-01-20 (cit. on p. 9).

[90] K. P. Sinaga and M.-S. Yang. "Unsupervised K-Means Clustering Algorithm". In: *IEEE Access* 8 (2020), pp. 80716–80727. DOI: 10.1109/ACCESS.2020.2988796 (cit. on pp. 30, 36, 45).

[91] *slowhttptest Usage Example*. https://www.kali.org/tools/slowhttptest/. Accessed: 2021-11-16 (cit. on p. 47).

[92] I. Syarif, A. Prugel-Bennett, and G. Wills. "Unsupervised Clustering Approach for Network Anomaly Detection". In: *Communications in Computer and Information Science*. Vol. 293 PART 1. 2012. DOI: 10.1007/978-3-642-30507-8_7 (cit. on p. 18).

[93] V. Teeraratchakarn and Y. Limpiyakorn. "Automated Monitoring and Behavior Analysis for Proactive Security Operations". In: *Proceedings of the 2020 2nd International Conference on Management Science and Industrial Engineering*. MSIE 2020. Osaka, Japan: Association for Computing Machinery, 2020, pp. 105–109. ISBN: 9781450377065. DOI: 10.1145/3396743.3396787. URL: https://doi.org/10.1145/3396743.3396787 (cit. on p. 23).

[94] *tshark*. https://www.wireshark.org/docs/man-pages/tshark.html. Accessed: 2021-03-20 (cit. on p. 28).

[95] R. Vaarandi, B. Blumbergs, and M. Kont. "An unsupervised framework for detecting anomalous messages from syslog log files". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, pp. 1–6. DOI: 10.1109/NOMS.2018.8406283 (cit. on p. 12).

[96] M. Wang, L. Xu, and L. Guo. "Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning". In: *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*. 2018, pp. 140–144. DOI: 10.1109/ICFSP.2018.8552075 (cit. on p. 12).

[97] *Web Security Academy - SQL injection*. https://portswigger.net/web-security/sql-injection. Accessed: 2021-11-20 (cit. on p. 48).

[98] *What Are Bot Attacks?* https://www.signalsciences.com/glossary/bot-attack-protection/. Accessed: 2021-11-20 (cit. on p. 49).

[99] R. Winter. "Design science research in Europe". In: *European Journal of Information Systems* 17.5 (2008), pp. 470–475. DOI: 10.1057/ejis.2008.44. URL: https://doi.org/10.1057/ejis.2008.44 (cit. on p. 3).

[100] *Wireshark · Go Deep*. https://www.wireshark.org/. Accessed: 2021-03-20 (cit. on pp. 13, 28, 40).

[101] *XAMPP*. https://www.apachefriends.org/index.html. Accessed: 2021-10-20 (cit. on p. 40).

[102] Y. Xin et al. "Machine Learning and Deep Learning Methods for Cybersecurity". In: *IEEE Access* 6 (2018), pp. 35365–35381. DOI: 10.1109/ACCESS.2018.2836950 (cit. on pp. 17, 20, 30).

[103] C. Yin et al. "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks". In: *IEEE Access* 5 (2017), pp. 21954–21961. DOI: 10.1109/ACCESS.2017.2762418 (cit. on p. 20).

[104] *Z-Score and How It's Used to Determine an Outlier*. https://medium.com/clarusway/z-score-and-how-its-used-to-determine-an-outlier-64210f3b482. Accessed: 2021-09-13 (cit. on p. 44).

[105] M. Zamani and M. Movahedi. *Machine Learning Techniques for Intrusion Detection*. 2015. arXiv: 1312.2177 [cs.CR] (cit. on pp. 18, 19).

I

ANNEXES

Feature Name	Description
Protocol	Protocol used
Flow duration	Duration of the flow in Microsecond
Total Fwd Packet	Total packets in the forward direction
Total Bwd packets	Total packets in the backward direction
Total Length of Fwd Packet	Total size of packet in forward direction
Total Length of Bwd Packet	Total size of packet in backward direction
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Max	Maximum size of packet in forward direction
Fwd Packet Length Mean	Mean size of packet in forward direction
Fwd Packet Length Std	Standard deviation size of packet in forward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Bwd Packet Length Std	Standard deviation size of packet in backward direction
Flow IAT Mean	Mean time between two packets sent in the flow
Flow IAT Std	Standard deviation time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction

Figure I.1: Attack Detection features provided by *CICFlowMeter* tool [30] (Part 1).

Feature Name	Description
Fwd IAT Total	Total time between two packets sent in the forward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Fwd PSH flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the backward direction
Fwd Packets/s	Number of forward packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a packet
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH

Figure I.2: Attack Detection features provided by *CICFlowMeter* tool (Part 2).

Feature Name	Description
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWR Flag Count	Number of packets with CWR
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet Size	Average size of packet
Fwd Segment Size Avg	Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in the backward direction
Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
Fwd Init Win bytes	The total number of bytes sent in initial window in the forward direction
Bwd Init Win bytes	The total number of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active

Figure I.3: Attack Detection features provided by *CICFlowMeter* tool (Part 3).

Feature	Type
cpu system	double
cpu iowait	double
cpu idle	double
cpu user	double
cpu softirq	double
load 1 min	double
load 5 min	double
load 15 min	double
memory used pct	double
memory actual used pct	double
diskio io ops	double
diskio iostat queue avg size	double
diskio iostat request avg size	double
diskio iostat await	double
diskio iostat service time	double
diskio iostat busy	double
diskio iostat read await	double
diskio iostat read request merges per sec	double
diskio iostat read request per sec	double
diskio iostat read per sec bytes	double
diskio iostat write await	double
diskio iostat write request merges per sec	double
diskio iostat write request per sec	double
diskio iostat write per sec bytes	double
processes unknown	integer
processes total	integer
processes sleeping	integer
processes running	integer
processes idle	integer
labor time	boolean
week	boolean

Figure I.4: Anomaly Detection features provided by *Metricbeat* agent [14] on the *"Storage"* and *"Sql"* servers.

Feature	Type
cpu system	double
cpu iowait	double
cpu user	double
cpu softirq	double
load 1 min	double
load 5 min	double
load 15 min	double
memory used pct	double
memory actual used pct	double
sda diskio io ops	double
sda diskio iostat queue avg size	double
sda diskio iostat request avg size	double
sda diskio iostat await	double
sda diskio iostat service time	double
sda diskio iostat busy	double
sda diskio iostat read await	double
sda diskio iostat read request merges per sec	double
sda diskio iostat read request per sec	double
sda diskio iostat read per sec bytes	double
sda diskio iostat write await	double
sda diskio iostat write request merges per sec	double
sda diskio iostat write request per sec	double
sda diskio iostat write per sec bytes	double
sdb diskio io ops	double
sdb diskio iostat queue avg size	double
sdb diskio iostat request avg size	double
sdb diskio iostat await	double
sdb diskio iostat service time	double

Figure I.5: Anomaly Detection features provided by *Metricbeat* agent on the *"App"* server (Part 1).

Feature	Type
sdb diskio iostat busy	double
sdb diskio iostat read await	double
sdb diskio iostat read request per sec	double
sdb diskio iostat read per sec bytes	double
sdb diskio iostat write await	double
sdb diskio iostat write request merges per sec	double
sdb diskio iostat write request per sec	double
sdb diskio iostat write per sec bytes	double
sdc diskio io ops	double
sdc diskio iostat queue avg size	double
sdc diskio iostat request avg size	double
sdc diskio iostat await	double
sdc diskio iostat service time	double
sdc diskio iostat busy	double
sdc diskio iostat read await	double
sdc diskio iostat read request merges per sec	double
sdc diskio iostat read request per sec	double
sdc diskio iostat read per sec bytes	double
sdc diskio iostat write await	double
sdc diskio iostat write request merges per sec	double
sdc diskio iostat write request per sec	double
sdc diskio iostat write per sec bytes	double
processes unknown	integer
processes total	integer
processes sleeping	integer
processes running	integer
processes idle	integer
labor time	boolean
week	boolean

Figure I.6: Anomaly Detection features provided by *Metricbeat* agent on the *"App"* server (Part 2).