



ΣΤΡΑΤΙΩΤΙΚΗ ΣΧΟΛΗ ΕΥΕΛΠΙΔΩΝ
Τμήμα Στρατιωτικών Επιστημών

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΔΙΔΡΥΜΑΤΙΚΟ ΔΙΑΤΜΗΜΑΤΙΚΟ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΑΚΑΔΗΜΑΪΚΟΥ ΕΤΟΥΣ 2016-17
ΣΧΕΔΙΑΣΗ & ΕΠΕΞΕΡΓΑΣΙΑ
ΣΥΣΤΗΜΑΤΩΝ (SYSTEMS ENGINEERING)

(ΠΔ 96 /2015/ΦΕΚ 163Α'/20.08.2014)



ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Σχολή Μηχανικών Παραγωγής & Διοίκησης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Σχεδίαση και ανάπτυξη γνωστικής
Μηχανής Ανάλυσης Δεδομένων για δικτυακή
ασφάλεια, με εφαρμογή τεχνολογιών Big Data
και μεθόδων μηχανικής μάθησης

Διατριβή που υπεβλήθη για την μερική ικανοποίηση των απαιτήσεων για την
απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης

Υπό τον:

ΠΑΠΑΔΟΠΟΥΛΟ ΔΗΜΗΤΡΙΟ

A.M.: 2015018012

ΣΕΠΤΕΜΒΡΙΟΣ 2017



HELLENIC ARMY ACADEMY
Department of Military Science

HELLENIC REPUBLIC
INTER-FACULTY INTER-DEPARTMENTAL
POSTGRADUATE PROGRAMME OF THE
ACADEMIC YEAR 2016-17
SYSTEMS ENGINEERING

(PD 96 /2015/GG 163A'/20.08.2014)



TECHNICAL UNIVERSITY OF CRETE
School of Production Engineering and
Management

MASTER'S THESIS

Design and development of a cognitive
Data Analytics Engine for network security,
implementing Big Data technologies &
machine learning techniques

Thesis submitted in partial fulfilment of the requirements for the
degree of Master of Science in Systems Engineering

By:

Papadopoulos Dimitrios

S.R.N.: 2015018012

SEPTEMBER 2017

The Master's Thesis of Dimitrios Papadopoulos is approved by:

THREE-MEMBER EXAMINATION BOARD:

Professor (Supervisor) Nikolaos Papadakis

Professor Constantinos Karamatsoukis

Professor Nikolaos Spanoudakis

© *Copyright by Dimitrios Papadopoulos*

Year 2017

*This work is licensed under a Creative Commons Attribution-ShareAlike 4.0
International License.*



*Lust for victory will not give you the victory.
You must receive the victory from your opponent.
His movement will be unnatural,
so you will be able to know what he is going to do before he does it.*

Soke Masaaki Hatsumi

Page intentionally left blank

Acknowledgments

I would like to express my gratitude to my supervisor Assistant Professor Nikolaos Papadakis for the useful comments, remarks and engagement throughout the learning process of this master's thesis. His flexibility, genuine caring and faith in me from the beginning of our cooperation helped me to delve into a new sea of knowledge without feeling drowned.

My gratitude is also extended to all the partners of the SHIELD consortium with whom I've had the pleasure to work with actively in this project and gain extensive professional guidance, looking upon them as role models during my first steps in scientific research.

This work would not have been possible without the technical help of the Apache Spot community during the implementation phase of the project. Their friendly, direct and supportive nature helped me overcome all the obstacles that arose from the complexity of this venture.

Nobody has been more important to me in the pursuit of this thesis than my loved ones. I would like to thank my parents, whose love and guidance are with me in my every accomplishment. A heartfelt thanks to my loving and supportive girlfriend Katerina, whose passion for the subject was the spark that triggered mine as well. Her extensive help and devotion to my cause was the best encouragement I could possibly wish for.

Dimitris

Page intentionally left blank

Table of Contents

Abstract	1
Abstract in Greek	2
Introduction	3
Chapter 1	9
Big Data in Network Security	
1.1 Introduction to Big Data	9
1.2 Definitions and features of Big Data	10
1.3 The impact of Big Data on modern applications	14
1.4 Challenges of Big Data and comparison with similar technologies	24
1.5 The current landscape of network security and the necessity of new solutions	27
1.6 Network analytics in the era of Big Data	32
Chapter 2	37
Machine Learning in Cybersecurity	
2.1 Addressing the vulnerabilities of cyber infrastructure	37
2.2 Traditional and unconventional approaches to cybersecurity	39
2.2.1 Signature-based detection systems	40
2.2.2 Anomaly detection systems	43
2.3 Machine learning methods in anomaly detection systems	45
2.3.1 Supervised machine learning methods in cybersecurity	47
2.3.1.1 Artificial Neural Networks	47
2.3.1.2 Support Vector Machines	49
2.3.1.3 Decision Trees	51

2.3.1.4 Hidden Markov Model	53
2.3.2 Unsupervised machine learning methods in cybersecurity	55
2.3.2.1 k-Means Clustering	57
2.3.2.2 Expectation Maximization	57
2.3.2.3 k-Nearest Neighbour	58
2.3.2.4 Latent Dirichlet Allocation (LDA)	60
Chapter 3	
Designing a Cybersecurity Engine	67
3.1 Performing anomaly detection in cybersecurity engines	67
3.2 Profile of an anomaly detection engine	69
3.3 Overview of the DARE, a modern IDPS based on machine-learning	71
3.4 Architecture of the DARE	75
3.4.1 The Data Acquisition and Preparation phase	78
3.4.2 The Data Analytics Phase	81
3.4.2.1 Cognitive Data Analytics module	82
3.4.2.2 Security Data Analysis module	84
3.4.2.3 The procedure from data acquisition to anomaly detection	86
3.4.3 The Remediation Phase	88
Chapter 4	
Implementation of the Cybersecurity Analytics Engine	93
4.1 Specifications and used technologies	93
4.2 Implementation of the data ingestion process	95
4.3 Implementation of the data analytics process	98
4.3.1 Latent Dirichlet Allocation implementation for network anomaly detection	100

4.3.2	Configuration of the anomaly detection framework	108
4.4	Example use of the cybersecurity engine	112
4.4.1	Acquisition and ingestion of netflow data	112
4.4.2	Machine learning for netflow data	114
4.4.3	Operational analytics and visualisation for netflow data	116
Chapter 5		119
Anomaly Detection Testing and Findings		
5.1	Description of test cases	119
5.2	Analysis on publicly available datasets	120
5.2.1	Test on netflow data	120
5.2.2	Test on DNS data	122
5.2.3	Test on proxy data	125
5.3	Analysis on captured data	126
5.3.1	DoS detection using the netflow pipeline	127
5.3.2	Anomaly detection in enterprise-level network traffic	130
Chapter 6		137
Conclusions		
6.1	Concluding remarks	137
6.2	Future work	141
References		143
List of Acronyms		149
Appendix		151
	Apache Spot Installation Procedure	151

List of Figures

Chapter 1: Big Data in Network Security

Figure 1.1	The “5Vs” of Big Data	12
Figure 1.2	Gartner’s Hype Cycles 2012-2013	15

Chapter 2: Machine learning in Cybersecurity

Figure 2.1	Workflow of a typical signature detection system	42
Figure 2.2	Machine learning methods taxonomy	46
Figure 2.3	Example of a two-layer ANN framework	48
Figure 2.4	Hyperplane and support vectors of a linear SVM algorithm	50
Figure 2.5	Structure of a binary decision tree	52
Figure 2.6	Hidden states and observations of a Hidden Markov Model algorithm	53
Figure 2.7	k-means clustering example	56
Figure 2.8	Convergence of an EM procedure	58
Figure 2.9	k-nearest neighbour example, k=5	59
Figure 2.10	Graphical model representation of LDA	63

Chapter 3: Designing a cybersecurity engine

Figure 3.1	Workflow of an anomaly detection system	70
Figure 3.2	Overview of the SHIELD platform	72
Figure 3.3	Dataflow diagram between the DARE and other SHIELD components	76
Figure 3.4	The three phases of the DARE platform	77
Figure 3.5	Centralised low-level ingestion architecture	79
Figure 3.6	Distributed low-level ingestion architecture	79
Figure 3.7	Data Analytics Framework overview	81
Figure 3.8	Machine learning entity overview	83
Figure 3.9	Operational analytics subcomponent overview	84
Figure 3.10	Security Data Analysis module overview	85
Figure 3.11	Flow diagram of the data acquisition phase	87
Figure 3.12	Flow diagram of the anomaly detection phase	87
Figure 3.13	The recommendation and remediation subcomponents	89

Chapter 4: Implementation of the cybersecurity analytics engine

Figure 4.1	List of subcomponents and used technologies	95
Figure 4.2	Dataflow of the ingestion framework	96
Figure 4.3	Recommended node configuration of the CDH cluster	109
Figure 4.4	Roles of the cluster nodes after the CDH configuration	111
Figure 4.5	The Cloudera Manager Interface	111
Figure 4.6	Example of the nohup.out log	114
Figure 4.7	Example of the execution of the anomaly detection script on netflow data	115
Figure 4.8	Example of the anomaly detection .csv file generated by the machine-learning node	116
Figure 4.9	Suspicious analysis view of the Spot browser-based GUI	117
Figure 4.10	Ingest summary view of the Spot browser-based GUI for netflow ingestion	118

Chapter 5: Anomaly detection testing and findings

Figure 5.1	View of the analysis results on netflow data	122
Figure 5.2	View of the analysis results on DNS data	124
Figure 5.3	View of the analysis results on proxy data	126
Figure 5.4	View of the DoS detection results	128
Figure 5.5	View of the Distributed DoS detection results	130

Appendix: Apache Spot Installation Procedure

Figure A.1	Example of spot.conf	153
Figure A.2	Example of hdfs_setup.sh script	155
Figure A.3	The ingest_conf.json file	157
Figure A.4	Engine.json configuration	159

List of Tables

Chapter 4: Implementation of the cybersecurity analytics engine

Table 4.1	Probability distributions used by LDA	102
	Correspondence between text corpora as seen on LDA	
Table 4.2	mathematical principles and network logs used by Spot's LDA implementation	102
Table 4.3	Word creation for netflow logs	104
Table 4.4	Word creation for DNS logs	106
Table 4.5	Word creation for proxy logs	107

Chapter 5: Anomaly detection testing and findings

Table 5.1	DoS attack specifications	128
Table 5.2	Distributed DoS attack specifications	129
Table 5.3	Ranking of attacks performed by BreakingPoint	135

Abstract

The main objectives of this Master's thesis may be summarized as follows:

- a. The presentation of Big Data's impact on modern applications, with particular emphasis on cybersecurity analytics. The review of the challenging aspects as an aftermath of the world's transformation towards a data-driven culture and the identification of the arisen opportunities in the field of network security .
- b. The study of machine learning utilisation in cybersecurity for the extraction of hidden knowledge in accumulated network data and the comparison between cognitive anomaly detection systems and traditional signature-based systems. The categorisation of machine-learning methods to supervised and unsupervised and the presentation of the mathematical background regarding the most common algorithms of each family, along with several cybersecurity implementations .
- c. The architectural design, development and implementation of a state-of-the-art data analytics engine, in the framework of the SHIELD EU-funded cybersecurity project. The presentation of all the relevant components, placing more focus on the description of the data acquisition and data analytics modules which constitute the core of the platform.
- d. The deployment, configuration, usage and testing of the Apache Spot platform as an integrated analytics ecosystem for the accomplishment of anomaly detection, using public and captured network traffic datasets. The drawing of conclusions that identify the strong and weak points of the engine and can be generalised for the majority of cognitive analytics systems.

Περίληψη

Τα κύρια αντικείμενα που αναπτύχθηκαν στο πλαίσιο της παρούσας Μεταπτυχιακής Διατριβής αφορούν συνοπτικά:

- α. Την παρουσίαση της επίδρασης του Big Data στις σύγχρονες εφαρμογές, με ιδιαίτερη έμφαση στις αναλυτικές μεθόδους για την ασφάλεια του κυβερνοχώρου. Την ανασκόπηση των επακόλουθων προκλήσεων από τη μετάβαση σε έναν πολιτισμό οδηγούμενο από την πληροφορία και τον εντοπισμό των ευκαιριών που δημιουργούνται στον τομέα της δικτυακής ασφάλειας.
- β. Την μελέτη της χρήσης μηχανικής μάθησης σε εφαρμογές κυβερνοασφάλειας για εξαγωγή γνώσεων από συγκεντρωμένα δικτυακά δεδομένα και τη σύγκριση των σχετικών γνωστικών συστημάτων ανίχνευσης ανωμαλιών με παραδοσιακά συστήματα βασιζόμενα σε κανόνες. Την κατηγοριοποίηση των μεθόδων μηχανικής μάθησης σε μεθόδους μάθησης με επίβλεψη και χωρίς επίβλεψη και την παρουσίαση του μαθηματικού υποβάθρου των πιο γνωστών αλγορίθμων κάθε οικογένειας, συνοδευόμενη από εφαρμογές στον τομέα της ασφάλειας του διαδικτύου.
- γ. Τον αρχιτεκτονικό σχεδιασμό, την ανάπτυξη και υλοποίηση ενός σύγχρονου μηχανισμού ανάλυσης δεδομένων, στα πλαίσια του έργου SHIELD που χρηματοδοτείται από την ΕΕ για ασφάλεια στον κυβερνοχώρο. Την παρουσίαση όλων των σχετικών συναποτελούμενων τμημάτων, με έμφαση στην περιγραφή των τμημάτων απόκτησης και ανάλυσης δεδομένων που αποτελούν τον πυρήνα της πλατφόρμας.
- δ. Την ανάπτυξη σε δοκιμαστικό περιβάλλον, διαμόρφωση, χρήση και δοκιμή της πλατφόρμας Apache Spot ως ένα ολοκληρωμένο οικοσύστημα ανάλυσης δεδομένων για την ανίχνευση ανωμαλιών, χρησιμοποιώντας δημοσίως διαθέσιμα καθώς και συλλεγμένα σύνολα δικτυακών δεδομένων. Την εξαγωγή συμπερασμάτων που αφορούν τα ισχυρά και αδύνατα σημεία της πλατφόρμας, αλλά μπορούν να γενικευτούν ώστε να αφορούν την πλειονότητα των γνωστικών συστημάτων ανάλυσης.

Introduction

The growing prevalence of cyber threats in the world has led to the deployment of various security monitoring systems to protect the network and its resources from cyberattacks. Conventional approaches to cybersecurity leverage firewalls, authentication tools, and other rule-based systems that monitor, track, and block malicious attacks. These approaches create a fragile protective shield -however- since their vulnerabilities are ubiquitous, because of the flawed design and implementation of software and network infrastructure. Although patches are constantly being developed to eliminate these vulnerabilities, attackers are continuously exploiting newly discovered flaws and more complex penetration methods. Because of the constantly evolving cyber threats, building rule-based systems for discovered attacks is not enough to protect users. Higher-level methodologies based on cognitive analytics, able to exploit the hidden information in massive amounts of network data in order to discover hidden patterns of malicious behaviour are also required to discover the lurking intrusion techniques, so that a more reliable security cyber infrastructure can be materialized.

This thesis presents part of the work done within the framework of the SHIELD¹ EU-funded Horizon 2020 project. It pertains to the design, development and implementation of a cybersecurity data analytics engine that relies on machine-learning techniques to detect and mitigate network anomalies by exploiting traffic logs acquired from enterprise-level networks and Internet Service Providers (ISPs) and with the help of virtualisation technologies. The idea of this project is to use machine learning algorithms applied to network traffic to offer greater security to the services that are currently provided, mainly

¹ SHIELD, Securing against intruders and other threats through a NFV-enabled environment: http://cordis.europa.eu/project/rcn/202684_en.html

at the corporate context. A catalogue of virtualized network security functions (vNSFs) allows for the collection of traffic and mitigation of attacks in the network, based on the analytics, decisions and recommendations of a cognitive system, the Data Analysis and Remediation Engine (DARE) that acts through an orchestrator for virtualized environments.

The thesis focuses mainly on the implementation of cognitive analytics for anomaly detection, providing the necessary background first to help the reader understand the most important concepts of Big Data and machine learning in cybersecurity, as well as the basic principles of modern intrusion detection and prevention systems (IDPSs).

The main body of this thesis is organised into 6 chapters. Each chapter is divided into sections and some sections are further divided into subsections for better readability. An Appendix containing technical information about the deployed analytics engine is also included at the end. Next follows a brief presentation of each chapter:

- ❖ **Chapter 1** presents the main features that characterise Big Data and tries to map their impact on modern applications. The challenges that occur from the acquisition, storage and processing of enormous amounts of data and the transformative effect that they induce on many aspects of a business are described, focusing mainly on the field of network analytics. The usefulness of exploiting Big Data in cybersecurity to detect previously unprecedented threats is explained with the depiction of industry practices that rely on Big Data to discover hidden patterns of anomalous behaviour, which were so far impossible to detect with the use of traditional tools.
- ❖ In **Chapter 2**, the use of machine learning in network security to exploit the hidden information in Big Data is presented. A comparison between cognitive anomaly detection systems based on machine-learning and traditional signature-based systems is done to showcase the superiority of the former in the detection of increasingly complex and unique threats. The categorisation of machine-learning methods to supervised and unsupervised is thoroughly explained and the

mathematical background of the most common algorithms of each family is presented, along with several cybersecurity implementations, that served as both the inspiration and the foundation of the deployed analytics engine.

- ❖ **Chapter 3** compares a typical cognitive anomaly detection system based on modern bibliography, with the architectural design of the developed data analytics engine in the framework of the SHIELD project. A detailed view of all the modules that compose the engine is given, though more focus is placed on the description of the data acquisition and data analytics phases which compose the core of the platform as well as the main scope of this thesis
- ❖ **Chapter 4** presents the main technology that is used to implement the above phases concentrated in the Apache Spot platform, as well as the algorithmic implementation of the utilised topic modelling method for network anomaly detection. The use of the engine is further demonstrated via an example that depicts all the stages from the collection of data to the presentation of results.
- ❖ **Chapter 5** includes several tests performed in the testbed network of the deployed anomaly detection engine with both publicly available datasets and captured network traffic that contain logs of malicious activity. The tests indicate the strong and weak points of the engine's capabilities and lead to the extraction of useful conclusions.
- ❖ Finally, **Chapter 6** lists all the conclusions of the work presented in this thesis, mainly regarding the efficiency of the anomaly detection algorithm. It also suggests relevant future work that could be done in the domain in order to further exploit the potential of the utilised framework by extending its detection capabilities and optimising the operation of its components towards real-time detection and threat mitigation.

Chapter 1

Big Data in Network Security

1.1 Introduction to Big Data

Over the past 20 years, data has increased in a large scale in various fields. Big Data has become one of the most widely used terms in information technology world, but the implications and major benefits for those investing in it are yet to be realized in the coming years. According to IBM, $2.5 \cdot 10^5$ bytes of data were being created daily in 2013, which, in relative terms, means that 90% of the data in the world had been created in the last two years (IBM, 2013). Gartner had successfully projected that by 2015, 85% of Fortune 500 organizations would be unable to exploit Big Data for competitive advantage and about 4.4 million jobs would be created around Big Data (Gartner, 2012). Despite the fact that these estimates should not be interpreted in an absolute sense, their confirmation is a strong indication of the ubiquity of Big Data and the strong need for analytical skills and resources because, as more data is being accumulated, managing and analysing these data resources in the most optimal way become critical success factors in creating competitive advantage and strategic leverage.

The term of Big Data was coined under the explosive increase of global data and was mainly used to describe these enormous datasets. Compared with traditional datasets, Big Data generally includes masses of unstructured, semi-structured or tabular data that mainly require real-time analysis. In addition, Big Data also brings new opportunities for discovering new values, as it allows for an in-depth understanding of the hidden values while it incurs new challenges, e.g., on how to effectively organize and manage such data.

At present, Big Data has attracted considerable interest from industry, academia, and government agencies.

The global data explosion is highly driven by technologies including digital video, music, smartphones, and the Internet (Oracle, 2012). This data has its origins in a variety of sources, including web searches, sensors, commercial transactions, social media interactions, video uploads, and mobile phone GPS signals.

The recent rapid growth of Big Data is mainly a result of people's daily lives, especially related to the service of Internet companies. It is estimated that every minute email users send more than 204 million messages, Google receives over 2.500.000 search queries, Facebook users share more than 684.000 pieces of content, Twitter users send over 350.000 tweets, more than 570 new websites are created, and Instagram users share 66.000 new photos. All these sources of information will contribute to reach 35 Zettabytes of data stored by 2020 (Eaton et. al, 2012).

The rapid growth of cloud computing and the Internet of Things (IoT) further promote the sharp growth of data. Cloud computing provides safeguarding, access sites, and channels for data asset. As a result of this tremendous IoT expansion, sensors all over the world are collecting and transmitting data of many different types that will be stored and processed in the cloud. Such data in both quantity and variety is expected to far surpass the capacities of the IT architectures and infrastructure of existing enterprises, and its real-time requirement is going to greatly stress any available computing capacity. (Chen et al., 2014)

1.2 Definitions and features of Big Data

The concept of Big Data is an abstract one, as it also has some hidden features that determine the difference between itself and just "large amounts of data". At present, although the importance of Big Data has been generally recognized, people still have

different opinions on its definition. In general, Big Data refers to the datasets that could not be perceived, acquired, managed, and processed by traditional IT and software/hardware tools within a tolerable time (Kumar et al., 2016). Different applications have led scientific and technological enterprises, research scholars, data analysts, and technical practitioners to adopt different definitions of Big Data. The most concise definition is perhaps, the one from the US National Institute of Standards and Technology (NIST, 2013) which defines Big Data as “the data of which the volume, acquisition speed and data representation limits the capacity of using traditional relational methods to conduct effective analysis or the data which may be effectively processed with important horizontal zoom technologies”. NIST focuses on the technological aspect of Big Data and indicates that efficient methods or technologies need to be developed and used to analyse and process Big Data.

While it is undeniable that the amount and variety of sources of information have created significant storage, processing and analysis challenges for the effective use of Big Data, its tremendous size and complexity are only one side of the issue. The other aspects are the demands for cost effective forms of capture, storage, analytics and visualization. All these aspects can be better depicted by the five most important features or dimensions that characterize Big Data, commonly referred to as the “5Vs” (Waterman et al, 2015). These namely are (Figure 1.1):

- **Volume** - The quantity of data.
- **Velocity** - The speed/rate at which data is provided.
- **Variety** - The different formats and semantic models in which data is provided.
- **Veracity** - The trustworthiness and accountability of data.
- **Value** - The usefulness of the acquired data.

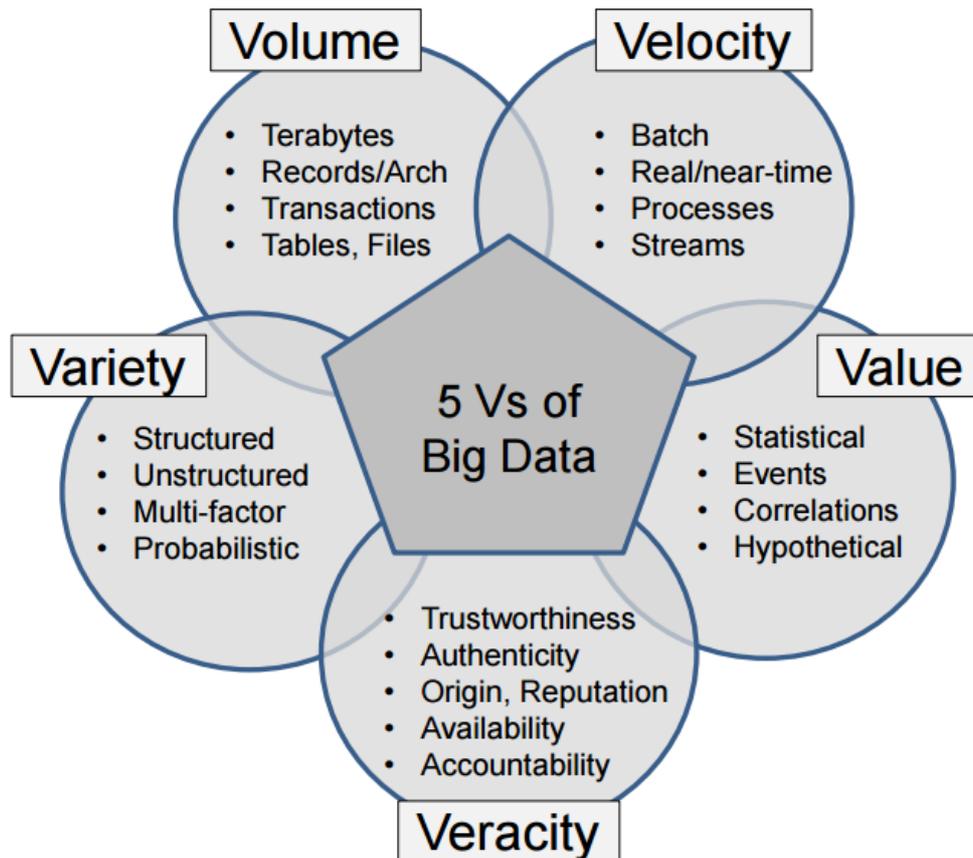


Figure 1.1: The “5Vs” of Big Data

Volume

Volume is the most challenging aspect of Big Data since it imposes a need for scalable storage and a distributed approach to querying. Big enterprises already have a large amount of data accumulated and archived over the years that could be in the form of system logs, record keeping etc. The amount of this data easily gets to the point where conventional database management systems may not be able to handle it. Data warehouse based solutions may not necessarily have the ability to process and analyse this data, due to lack of parallel processing architecture. At the same moment, email communication patterns, consumer preferences and trends in transaction-based data can be derived from text data,

locations or log files. Big Data technologies offer solutions to create value from this massive and previously unused or difficult to process data.

Velocity

Data is flowing into our everyday lives at a large speed as Web and mobile technologies have enabled generating a data flow back to the providers. Online shopping has revolutionized consumer and provider interactions. Online retailers can now keep log of and have access to customers' every interaction and need fast reactions in order to utilize this information in recommending products, thus putting their organization on a leading edge. Online marketing organizations are deriving lot of advantage with the ability to gain insights instantaneously. The invention of the smartphone era has led to even further location based data being generated, signifying the importance of being able to take advantage of this huge amount of data in a limited amount of time.

Variety

Data that are being generated with social and digital media are rarely structured. Unstructured text documents, video, audio data, images, financial transactions, interactions on social websites are examples of unstructured data. While conventional databases are able to support "large objects" (LOBs), they are bound to their limitations if not distributed. These datasets are hard to fit in conventional relational database management structures and is not very integration-friendly, ultimately leading to loss of information. Big Data structures, on the other hand, tend to keep all the data based on the assumption that insights can be hidden in every bit.

Veracity

Having a lot of data in different volumes coming in at high speed is worthless if that data is incorrect, either by being untrue or corrupted. Incorrect data is an inevitable phenomenon that causes a lot of problems mainly for organisations, but for consumers as well. Therefore, it is crucial to ensure that both the data and the analyses performed on that data are correct, especially in automated decision-making situations, where no human is involved.

Value

Value represents the outcome to be derived from Big Data analysis, the translation of both structured and unstructured data into insights that will lead to provide correct recommendations, generate business revenue and provide cost-savings, thus resulting to a competitive advantage. Users of Big Data have to identify these variables that can affect their processes the most in terms of expected outcome and focus on how these interact with real-time predictability and mobility, in order to provide a strategic overview of their goal.

1.3 The impact of Big Data on modern applications

Over the past few years, nearly all major companies, including EMC, Oracle, IBM, Microsoft, Google, Amazon, and Facebook, etc., have started their Big Data projects. Taking IBM as an example, it has invested, since 2005, USD 16 billion on 30 acquisitions related to Big Data (Chen et al., 2014). In academia, Big Data remains also under the spotlight, with a vast number of nationally and internationally funded research programs (e.g. H2020, FP7) focusing on extracting value from Big Data analytics. In the beginning of 2012, a report titled “Big Data, Big Impact” presented at the Davos Forum in

Switzerland, announced that Big Data has become a new kind of economic asset, just like currency or gold (WEF, 2012). Gartner issued Hype Cycles from 2012 to 2013, which classified Big Data computing, social analysis, and stored data analysis into 48 emerging technologies that deserve most attention as seen in Figure 1.2 (Gartner, 2013). Many national governments have also paid great attention to Big Data. In March 2012, the US Administration announced a USD 200 million investment to launch the Big Data Research and Development Initiative (White House, 2012). Four months later the Japan's ICT project issued by Ministry of Internal Affairs and Communications indicated that the Big Data development should be a national strategy and application technologies should be the focus. In July 2012, the United Nations issued "Big Data for Development" report, which summarized how governments utilized Big Data to better serve and protect their people (UN, 2012).

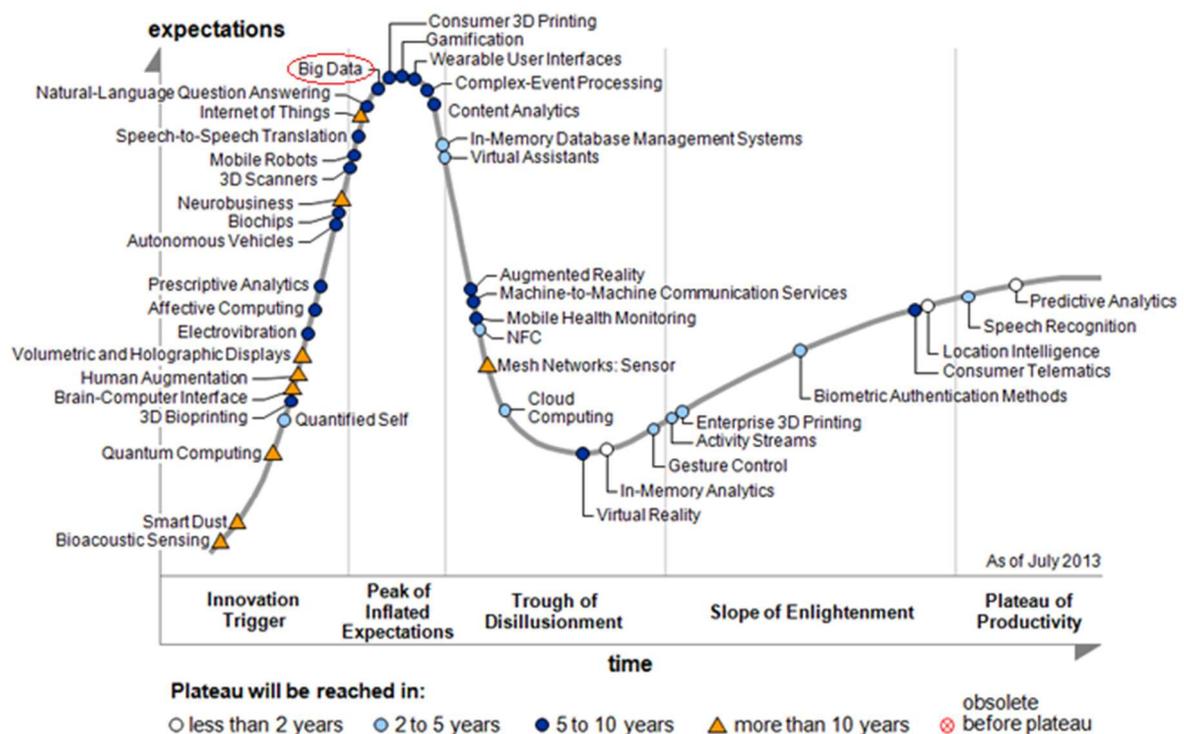


Figure 1.2: Gartner's Hype Cycles 2012-2013

The emergence of Big Data applications is apparent in various fields, some of them being more obvious than others (Srvanthi et al., 2015). Below are presented the most common applications where Big Data is involved:

- **Software:** Big Data have the potential to achieve valued insights for enhanced decision-making processes and have recently attracted substantial interest from both academics and practitioners (Sivarajah et al, 2017). Big Data Analytics (BDA) is increasingly becoming a trending practice that many organizations are adopting with the purpose of constructing valuable information from Big Data. The analytics process, including the deployment and use of BDA tools, is seen by organizations as a tool to improve operational efficiency, drive new revenue streams and gain competitive advantages over business rivals. Big Data Analytics Applications (BDA Apps) are a relatively new type of software applications, which analyse Big Data using massive parallel processing frameworks like Hadoop (Shang et al., 2013). Developers of such applications typically use a small sample of data in a pseudo-cloud environment. Afterwards, they deploy the applications in a large-scale cloud environment with considerably more processing power and larger input data. Big Data Analytics Applications are a new category of software applications that leverage largescale data, which is typically too large to fit in memory or even on one hard drive, to uncover actionable knowledge using large scale parallel-processing infrastructures. The Big Data can come from sources such as runtime information about news, tweets, stock market updates, usage information of an online game, or the data from any other rapidly growing data-intensive software system.
- **Banking:** Banks internationally are beginning to harness the power of data in order to derive utility across various spheres of their activity (Srivastava et al, 2015). Some of the most popular aspects where Big Data analytics is being successfully used in the banking sector are for discovering spending patterns, for customer segmentation

and profiling, for sentiment and feedback analysis and for fraud management. The accumulation of enormous amounts of customer data related to customers' bank accounts as well as the exchange of this information between organisations is common in modern countries and has invariably raised privacy issues. By uncovering hidden connections between seemingly unrelated pieces of data, Big Data analytics could potentially reveal sensitive personal information. It is indicated that most of the bankers are cautious in their use of Big Data due to privacy issues. Furthermore, outsourcing of data analysis activities or distribution of customer data across departments for the generation of richer insights also amplifies security risks. A recent security breach at a leading UK-based bank exposed databases of thousands of customer files (IBM, 2013). Although this bank launched an urgent investigation, files containing highly sensitive information such as customers' earnings, savings, mortgages, and insurance policies ended up in the wrong hands. Such incidents may reinforce concerns about data privacy and discourage customers from sharing personal information in exchange for customized offers.

- **Payment security:** Much like the other areas of finance, the payments industry is benefiting from adopting the latest techniques in data storage and analysis (Chemitiganti, 2017). Some of the ways that Big Data can be exploited to perform business functions include real-time risk scoring of payments, merchant analytics, money laundering detection etc. A modern example of Big Data utilisation is that of credit card companies, which rely on the speed and accuracy of in-database analytics to identify possible fraudulent transactions. By storing years' worth of usage data, they can flag atypical amounts, locations, and retailers, and follow up with cardholders before authorizing suspicious activity.
- **Enterprise:** Organizations are having difficulties in ingesting, storing, processing, transforming, and analyzing massive datasets. It has become clear that switching

successfully from a traditional storage to a Big Data storage approach can be a difficult task, especially with a growing number of new data sources and the need for increased processing capacity (Wolfson, 2015). In order to advance operational efficiencies and drive business growth, however, organizations must address and overcome these challenges. In recent years, many organizations have heavily invested in the development of enterprise data warehouses (EDW) to serve as the central data system for reporting, extract/transform/load (ETL) processes, and ways to take in data (data ingestion) from diverse databases and other sources both inside and outside the enterprise. In many industries, Big Data analytics are providing a competitive advantage; for example, data doesn't have to commute to work and back, thus it can deliver faster insights that help make informed decisions in real time for less expense than traditional data analysis tools.

- **Consumer Goods:** The impact of big data across the value chain in a consumer goods company affects many different levels such as marketing, sales, supply chain, finance and controlling and human relations (Khan, 2015). For instance, collecting consumer preference and purchasing data extracted from surveys, purchases, web logs, product reviews from online retailers, phone conversations with customer call centres, or even raw text picked up from around the Web, thus gathering tremendous information effortlessly, is a great convenience for the makers of consumer products. Their goal is to collect everything being communicated publicly about their products and extract meaning from it. By doing this, the company develops a nuanced understanding of why certain products succeed and why others fail. They can spot trends that can help them feature the right products in the right marketing media.
- **Agriculture:** Big Data is moving into agriculture as investors realize it can revolutionize the food chain from farm to table (Sparapani, 2017). Sensors on fields

and crops are starting to provide literally granular data points on soil conditions, as well as detailed info on wind, fertilizer requirements, water availability and pest infestations. GPS units on tractors, combines and trucks can help determine optimal usage of heavy equipment. Data analytics can help prevent spoilage by moving products faster and more efficiently. Unmanned aerial vehicles, or drones, can patrol fields and alert farmers to crop ripeness or potential problems. Biotechnology firms use sensor data to optimize crop efficiency. They plant test crops and run simulations to measure how plants react to various changes in condition. Their data environment constantly adjusts to changes in the attributes of various data it collects for each plant in the test bed. These simulations allow agriculture companies to discover the optimal environmental conditions for specific gene types.

- **Marketing:** In marketing, Big Data is providing insights into which content is the most effective at each stage of a sales cycle, how investments in Customer Relationship Management (CRM) systems can be improved, in addition to strategies for increasing conversion rates, prospect engagement, conversion rates, revenue and customer lifetime value. (Columbus, 2016) For cloud-based enterprise software companies, Big Data provides insights into how to lower the Customer Acquisition Cost (CAC), Customer Lifetime Value (CLTV), and manage many other customer-driven metrics essential to running a cloud-based business. Marketing organisations have also begun exploiting customer feedback by using facial recognition software to learn how well their advertising succeeds or fails at stimulating interest in their products. A recent study published in the Harvard Business Review looked at what kinds of advertisements compelled viewers to continue watching and what turned viewers off (Lerner et al., 2007). Among their tools was “a system that analyses facial expressions to reveal what viewers are feeling.” The research was designed to discover what kinds of promotions induced watchers to share the ads with their

social network, helping marketers create ads most likely to “go viral” and improve sales.

- **Telecommunications:** Service providers face an uphill challenge when they need to recommend new services to their customers, without overloading their networks and keeping their running costs under control. The market demands new set of data management and analysis capabilities that can help service providers make accurate decisions by taking into account customer, network context and other critical aspects of their businesses. Most of these decisions must be made in real time, placing additional pressure on the operators. Real-time predictive analytics exploiting Big Data can help leverage the information that resides in their multitude systems, make it immediately accessible and help correlate that data to generate insight that can help them drive their business forward. Telecommunication companies are mainly using Big Data to understand the potential of new product offerings, improve customer experiences, forecast network capacity, implement value-based network capacity planning, etc. (O'Brien, 2016).
- **Healthcare:** One of the most promising areas where big data can be applied to make a change is healthcare. Healthcare analytics have the potential to reduce costs of treatment, predict outbreaks of epidemics, avoid preventable diseases and improve the quality of life in general (Lebied, 2017). Although, the health care industry seems reluctant to the use of Big Data, mainly because of the doctors' resistance to change in making treatment decisions independently using their own clinical judgment rather than relying on protocols based on Big Data, there are numerous examples of Big Data in healthcare that prove that the development of medical applications of data should be the apple in the eye of data science, as they have the potential to save money and most importantly, people's lives. Despite the challenges regarding privacy concerns, lack of data management procedures and confidentiality issues,

healthcare stakeholders now have access to promising new threads of knowledge and are able to analyse Big Data in order to obtain useful insights. Although these efforts are still in their early stages, they could collectively help the industry address problems related to variability in health care quality and escalating health care spend. As recent technologic advances in the industry have improved their ability to work with healthcare data even though the files are enormous and often have different database structures and technical characteristics, they allow for the improvement of various services such as staff shift management, electronic health records, real-time alerting, predictive analytics, novel treatment research, telemedicine and sharing of medical information between hospitals.

- **Web services:** By becoming an increasingly digital society, the amount of web data being created and collected is growing and accelerating significantly. Analysis of this ever-growing data becomes a challenge with traditional analytical tools, thus new ways are required to bridge the gap between data being generated and data that can be analyzed effectively (Amazon Web Services, 2016). Several web sites are experiencing millions of unique visitors per day, each creating a large range of content. Companies increasingly want to be able to mine this data to understand limitations of their sites, improve response time and offer more target-based content. This requires tools to perform complicated analytics on data that far exceeds the memory of a single machine or even in cluster of machines. Big data tools and technologies offer opportunities and challenges in being able to analyse data efficiently to better understand customer preferences and gain a competitive advantage in the marketplace. More specifically, service oriented technologies like cloud computing deliver compute, storage and software applications as services over private or public networks based on subscription payment models.

- **Defence and military applications:** National security and defence related data being generated from multiple sources can be analysed for decision-making, military operations and better situational awareness, especially at national and joint services levels (Haridas, 2015). Today, machine data is generated by the movement of ships, aircraft and vehicles, satellites, drones, UAVs, reconnaissance aircrafts, sensors and radars at an astonishing rate. Moreover, human generated data from social media as well as business data generated from e-commerce transactions can be utilised for intelligence gathering, since inputs for national and military intelligence are obtained continuously during peace and war with the quantum increasing exponentially during crises and war. Since human analysis of this information and intelligence data is well beyond physical capability, Big Data analytics-based intelligence can provide the requisite output for decision-making and conduct of operations. Examples of modern Big Data applications for a military scope include the US Argus ground surveillance system that collects more than 40GB of information per second and Memex, developed by the US Defence Advanced Research Projects Agency (DARPA). Argus is an autonomous Real Time Ground Ubiquitous Surveillance Imaging System equipped with a 1.8 giga-pixel video camera with 12 frames per second (fps) and 368 sensors that collect 6,000 terabytes of data/imagery per day and feed it to Home Land Security. Memex, which stands for a combination of “memory” and “index” is an internet tool for internet crawling, searching, data aggregation, data analysis, data visualisation, data extraction and image analysis that analyses the data flowing in the US on a real time basis for proactive actions against terrorism and electronic warfare threats.
- **Cybersecurity:** Risk management and actionable intelligence is common in all Big Data analytics applications. The advantages of analysing large amounts of data to gain insights is undisputable, however the real added value comes from task

automation so that the data is available more quickly and the analysis is sent to the right people on time (CSO, 2016). This will allow analysts to classify and categorize cyber threats without the long delays that could make the data irrelevant to the attack at hand. Big Data can also help analysts to visualize cyberattacks by taking the complexity from various data sources and simplifying the patterns into visualizations. Being able to utilize the data in its raw format allows disparate data to be useful not only with what is happening currently, but also with past data. Using this past data, statistical baselines to identify what is normal and abnormal behaviour can be created, allowing us to determine when the data deviates from the norm. Although it may be difficult to catch all indicators when they are offered in real time, they may have a new meaning when they are viewed over time. This data can also create new possibilities for predictive models, statistical models, and machine learning that gives the ability to predict future events (Shukla, 2017). The application of Big Data analytics to network security is the key point of this dissertation and will be presented thoroughly in the following sections.

The next section describes the challenges met in procedures that involve Big Data and identifies its main differences with other similar technologies.

1.4 Challenges of Big Data and comparison with similar technologies

The sharply increasing data avalanche in the Big Data era brings huge challenges on its acquisition, storage, management and analysis. Traditional data management and analytics systems are based on the relational database management system (RDBMS). However, such RDBMSs only apply to structured data that has completely different properties in comparison with semi-structured or unstructured data. (Chen et al, 2014). In addition, RDBMSs are increasingly utilizing more and more expensive hardware. It is obvious that the traditional RDBMSs cannot handle the huge volume and heterogeneity of Big Data. The research community has proposed some solutions from different perspectives. For example, cloud computing is utilized to meet requirements on infrastructure for Big Data, such as cost efficiency, elasticity, and smooth upgrading/downgrading. For solutions of permanent storage and management of large-scale disordered datasets, distributed file systems and NoSQL databases are excellent choices (Howard et al. 1988). Such programming frameworks have achieved great success in processing clustered tasks, especially for webpage ranking. Various Big Data applications can be developed based on these innovative technologies or platforms. However, it is non-trivial to deploy the Big Data analytics systems. Some of the obstacles to be surpassed in the development of Big Data applications are listed as follows:

- **Data Representation:** Many datasets have certain levels of heterogeneity in type, structure, semantics, organization, granularity, and accessibility. Data representation aims to make data more meaningful for computer analysis and user interpretation. An improper data representation will reduce the value of the original data and may even obstruct effective data analysis. Efficient data representation shall reflect data

structure, class, and type, as well as integrated technologies, so as to enable efficient operations on different datasets.

- **Redundancy Reduction:** As there is a high level of redundancy in datasets, redundancy reduction is effective to reduce the indirect cost of the entire system on the premise that the potential values of the data are not affected. For example, most data generated by sensor networks are highly redundant, which may be filtered and compressed at several orders of magnitude.
- **Data Life Cycle Management:** Compared with the relatively slow advance of storage systems, pervasive computing is generating data at unprecedented rates and scales. Users of such systems are confronted with a lot of pressing challenges, one of which is that the current storage system could not support such massive data. As the values hidden in Big Data depend on data freshness, it is required to develop an importance principle related to the analytical value in order to decide which data shall be stored and which data shall be discarded.
- **Analytical Mechanism:** The analytical system of Big Data shall process masses of heterogeneous data within a limited time. However, traditional RDBMSs are strictly designed with a lack of scalability and expandability, which could not meet the performance requirements. Non-relational databases have shown their unique advantages in the processing of unstructured data and started to become mainstream in Big Data analysis. Even so, there are still some problems of non-relational databases in their performance and particular applications. A compromising solution needs to be found between RDBMSs and non-relational databases. Some enterprises like Facebook have already started utilising a mixed database architecture that integrates the advantages of both types of database. More research is also needed on the in-memory database and sample data based on approximate analysis.

- **Data Confidentiality:** Most Big Data service providers can't effectively maintain and analyse huge datasets because of their limited capacity. They must rely on professionals or tools to analyse the data, which increase the potential safety risks. For example, the transactional dataset generally includes a set of complete operating data to drive key business processes. Such data contains details of the lowest granularity and some sensitive information such as credit card numbers. Therefore, analysis of Big Data may be delivered to a third party for processing only when proper preventive measures are taken to protect the sensitive data, to ensure its safety.
- **Energy Management:** The energy consumption of mainframe computing systems has drawn much attention from both economy and environment perspectives. With the increase of data volume and analytical demands, the processing, storage, and transmission of Big Data will inevitably consume more and more electric energy. Therefore, system-level power consumption control and management mechanisms shall be established for Big Data while expandability and accessibility are both ensured.
- **Expendability and Scalability:** The analytical system of Big Data must support present and future datasets. The analytical algorithm must be able to process increasingly expanding and more complex datasets.
- **Cooperation:** The analysis of Big Data is an interdisciplinary research, which requires experts in different fields cooperate to harvest the potential of Big Data. A comprehensive Big Data network architecture must be established to help scientists and engineers in various fields access different kinds of data and fully utilize their expertise, so as to cooperate to complete the analytical objectives.

In the next sections of this chapter we will focus on the specific challenges that are encountered in the field of network security and how security analytics based on Big Data can help to effectively encounter these challenges. The background regarding the existing network security solutions is briefly presented in §1.5, while the ways these technologies can be enriched by the exploitation of Big Data are presented in §1.6.

1.5 The current landscape of network security and the necessity of new solutions

Network security refers to any activity designed to protect the usability and integrity of the users' network and data. It includes both hardware and software technologies such as access control policies, antivirus and antimalware software, behavioural analytics, data loss prevention, email security applications, firewalls etc. Effective network security targets a variety of threats and stops them from entering or spreading on the network. It has been made evident with numerous examples that most of the security strategies used in the past are increasingly less effective against new and complicated types of attacks. Many tools and security processes have been more focused on prevention than on detection and response, and attackers are taking advantage of the fact that organizations are not finding the indicators of compromise within their environments soon enough, nor are they responding to these incidents and removing them quickly enough.

Below are described the most prevalent technologies and tools that are leveraged in order to detect security breaches and are playing a significant role in incident response (SANS, 2016):

- **Network IDS/IPS (NIDS/NIPS):** Network intrusion detection and prevention systems are staples of network event generation that can often detect well-known

signatures of attacks or unusual patterns in network traffic. These platforms are exceedingly common in a defence-in-depth security architecture today, but they also generate a staggering array of alerts until properly tuned and are also prone to false positives. Signature-based NIDS/NIPS platforms are also limited to detection of known attacks.

- **Host IDS/IPS:** These applications can provide enormous amounts of useful data, especially when combating malware and attacks directed toward client-side software on endpoints. Much like network IDS/IPS, however, host-based IDS/IPS must be properly tuned to have true value for security analysts.
- **Security Information and Event Management (SIEM):** SIEM platforms can gather and correlate information from numerous devices and applications, allowing security analysts to monitor the environment more thoroughly and develop much more sophisticated detection rules. Many security operations centre (SOC) teams use SIEM tools as the primary monitoring and detection platform in the environment. Although SIEM tools are immensely powerful and provide advanced security monitoring and response features, they can also be complex to configure and manage.
- **Logging:** Centralized logging is a primary control for detecting security incidents today. Organizations increasingly work to gather and aggregate logs for analysis, with use cases ranging from IT operations troubleshooting to security event analysis. With this wealth of important data has come a significant challenge, however—many organizations are not able to wade through the data and successfully detect and prioritize the most meaningful events in their environments due to complexity of data involved, volume of data, lack of skill sets, and tools that do not facilitate rapid and intuitive searches. While logging and log analysis is a critical, and often required,

aspect of a security program, a fair amount of operational time is needed to properly develop analysis rules and alerts that benefit security teams.

- **Network device events:** Network devices such as firewalls, proxies, routers and switches can generate events in the form of standard syslog logs, vendor-specific event formats, and SNMP events. Access control rules, firewall rules and authentication logging can all provide important insight into what is happening on the network, as well as the configuration and management of the devices themselves. These devices play an important role in preventing attacks, as well as in detection of potential or existing adversaries looking to gain access into a network environment. As with general-purpose logging, however, the volume of network events is often overwhelming for today's security teams. It needs to be carefully analysed and sorted to properly weed out false positives and prioritize the most important security information.
- **Antivirus:** Antivirus tools are becoming less and less valuable over time, largely due to the system overhead used and the fact that many antivirus agents are focused primarily on signatures of well-known malware. Management and control of distributed agents on all endpoints can be an operational challenge as well. Aside from compliance requirements to have antivirus installed, security analysts may still find value in antivirus alerts, as they can indicate a broad array of malware infection attempts. In some cases, malware can be blocked or quarantined as well, giving incident responders time to properly contain the issue.
- **File Integrity Monitoring (FIM) and Whitelisting:** FIM and whitelisting are two additional host-based security controls that focus on changes to critical files and rules that allow only "known good" actions or content access by approved users. Both of these technologies can act as excellent prevention and detection controls, with valuable events generated and correlated with other system and environment

information. Once tuned and installed, both can add enormous value to detection and response programs. The biggest challenge in using FIM and whitelisting agents is the up-front requirements to properly configure and tune the policies necessary for proper functioning.

Despite the fact that all of the above detection and prevention tools can be effective against most types of threats, security teams are less effective than they could be because these disparate tools and platforms generate an overwhelming amount of data. Trying to incorporate numerous controls with detection events into response processes is becoming an increasingly difficult task and it can be easy to miss events and indicators of compromise. In addition, many teams tend to store a great deal of event data with the intention of analysing it later, which rarely happens. Many security teams are also using manual processes to initiate incident investigations and follow through with containment and elimination steps. This can be a slow process, leading to attackers moving laterally throughout many networks before investigators can detect, properly respond to and remediate intrusions. Overall, most organizations are operating reactively, trying to discover evidence of attacks and potential breaches from finite events using correlation tools such as SIEM and log management. However, the attacks today are much more subtle than before and may be detected only by looking into larger and longer-term patterns of behaviour in the IT environment.

Along with rapid event detection, correlation and response, the capability to predict future trends based on past and current behaviour is extremely important, which is where security analytics exploiting Big Data may prove useful. At the same time, having more data on which to perform root cause analysis and forensics can help security teams improve their ability to look for specific threats, thus leading to a more proactive monitoring approach. Security analytics examine large datasets with technologies that enable rapid and

accurate analysis, correlation and reporting to identify events and patterns of interest that may indicate malicious behaviour in the environment. With more data to work with, security teams can analyse the environment based on timing of events, sequences of occurrence, differences in data from various sources and real statistical analyses, like time series plots of risk and behaviour, machine learning, etc.

These security analytics systems need to perform rigorous analysis of many disparate types of data, and also provide strong correlation and statistical tools for security analysts to use in developing baselines of normal behaviour. In large networks, few tools are able to digest petabytes of network traffic, billions of network flows, and numerous other alerts and data types that can align with network data to determine whether unusual activities are occurring. In addition to the tactical detection and response capabilities, analytics tools should enable investigators to perform deep root cause analysis of incidents and develop predictive models of future behaviour based on knowledge of patterns in the data centre. All these can be achieved via methods of data analysis that automate analytical model building, using algorithms that iteratively learn from data and allowing the discovery of hidden insights without being explicitly aware of where to look, which is what machine learning does in a nutshell and will be presented thoroughly on the next chapter. Regardless of the individual datasets in use, analytics could provide predictive capabilities beyond what we've traditionally had with today's event management technology and may ideally lead to the development of baselines that allow the detection of new and unusual patterns for attacks both known and unknown.

1.6 Network analytics in the era of Big Data

It was made apparent from the previous paragraph that novel network security methods have to be implemented in order to increase the detection efficiency of the existing technologies. Big Data analytics can definitely play a major role in shifting cybersecurity systems from an exclusive rule-based approach to a more hybrid one that also encompasses machine learning methods in threat detection. However, this approach does not occur without its own challenges. Much of the challenge in performing security analytics stems from the irregular data that the analyst must handle (Talabis et al., 2014). There is no single standard data format or set of data definitions pertaining to data produced by computer systems and networks. For example, each server software package produces its own log file format. Additionally, these formats can generally be customized by users, which adds to the difficulty of building standard software tools for analysing the data. Another factor further complicating the analysis is that log files and other source data are usually produced in plain text format, rather than being organized into tables or columns. This can make it difficult or even impossible to import the data directly into familiar analytical tools. Additionally, security-related data is increasingly becoming too large to analyse with standard tools and methods. Large organizations may require multiple large data centres with an ever-growing collection of servers that are together by sprawling networks. All of this generates a huge volume of log files, which is essentially the one of the characteristics of Big Data.

As set out above, one of the driving forces behind the emergence of Big Data is the need for intelligence in making business decisions. Innovative technology is not the primary reason for the growth of the Big Data industry, in fact, many of the technologies used in data analysis, such as parallel and distributed processing, and analytics software and tools,

were already available. Changes in business practices (e.g., a shift to the cloud) and the application of techniques from other fields such as engineering, uncertainty analysis, behavioural science, etc. are what is driving the growth of data analytics. This emerging area created a new industry with experts (data scientists), who are able to examine and configure the different types of data into usable business intelligence.

In network security, many of the same analytical methods can be applied as well. These methods can be used to uncover relationships within data produced by servers and networks to reveal intrusions, denial of service attacks, attempts to install malware, or even fraudulent activity. Security analysis can range from simple observation by querying or visualizing the data, to applying sophisticated artificial intelligence applications. It can involve the use of simple spreadsheets on small samples of data, to applying Big Data, parallel-computing technologies to store, process and analyse terabytes, or even petabytes of data. Some analysis may only involve relatively small data sets, such as the instance in which a server has low traffic and only produces a single log file. However, data size can quickly increase, along with the computational power required for analysis when multiple servers are involved.

The sheer amount of data to review has led to one of the main challenges in network analytics, which is incident response. If done manually, the data that an incident responder would have to examine would be immense, potentially millions of lines of log information. Instead, by using Big Data techniques an incident responder will be able to combine many data sources with different structures together. Once that is completed, analytics techniques such as fuzzy searches, outlier detection, and time aggregations can be utilized to transform the data into more manageable datasets so that responders can focus their investigations on a smaller, more relevant subset of the data. Aside from logs, analytics techniques such as text analysis, can be useful in mining information from unstructured data sources. For

example, these techniques can be used to analyse security events from freeform text, providing insight into an organization's common security problems, or even finding security issues or incidents previously unknown.

Moreover, traditional ways to investigate or detect intrusions like using signatures or patterns can often be bypassed, depending on the type of the threat. For example, for an SQL injection attack, an incident responder will probably look for SQL statements in the logs, so he already knows what to look for. Those incidents that we know exist but we don't have any additional knowledge of them, are also called "Known Unknowns." Traditional security approaches usually work in this case, yet they do not cover the "Unknown Unknowns", which are the attacks that the incident responder has no knowledge of. Examples of this could be a zero-day attack or just something that the incident responder, or the investigative tool being utilized, is unfamiliar with or does not address. Typically, signature-based approaches are weak in detecting these types of attacks. Finding Unknown Unknowns are more in the realm of anomaly detection. For example, finding unusual spikes in traffic or outliers by using cluster analysis are good examples of analytics techniques that could potentially find incidents, which would otherwise have been missed by traditional means. It also helps in focusing the investigation to relevant areas, especially if there is a lot of data to sift through.

Cognitive analytics is a discipline that integrates human expertise into computer systems, aiming to automate or at least assist in manual decision making. If one can recreate the logic in identifying anomalous access events through cognitive analytics, the process of identifying them would be simpler, faster and can potentially be automated. By using Big Data analytics techniques such as clustering and visualization, organizations may be able to identify areas of "hotspots," thereby utilize resources more effectively and address vulnerabilities more systematically. Furthermore, another potentially interesting application

for network analytics is to predict future compromises based on previous compromises. For example, if a web server was hacked and the cause was unknown, analytics techniques such as machine learning could be used to profile the compromised server and to check if there are other similar servers that have the same profile. Servers with similar profiles would most likely be at risk of similar compromises and should be proactively protected.

In the following chapter, the role of machine learning in security analytics will be extensively presented. Special focus will be given to anomaly detection methods, covering the mathematical background of the most popular algorithms that are being utilised, along with implementation examples of each one, in the field of cybersecurity.

Chapter 2

Machine learning in Cybersecurity

2.1 Addressing the vulnerabilities of cyber infrastructure

In the past few years, cybercrime has continuously evolved its techniques to target victims, subvert critical data in information technology and exploit exposed devices to its attacks. It is expected that the emergence of the IoT (including mobility and heterogeneity of devices) as well as Big Data environments, will be two of the main targets of cybercrime in the years to come. An example of these new targets is that most of the devices involved on the massive and sustained Internet attack on October of 2016 were IoT devices (Kaspersky, 2016). Previously, the 2013 Norton Report had estimated economic losses due to consumer cybercrime - and for Europe alone - at 13 billion dollars (Paganini, 2013). Moreover, Ponemon study also points that the tendency of these economic losses is increasing (Ponemon, 2014).

The success of these attacks carries out too many negative consequences for the victims, where most of them are subjects to loss of sensitive data and intellectual property, opportunity costs (including service and employment disruptions), the damage to the brand image and company reputation, penalties and contractual compensations to customers of commercial networks after service disruptions, the cost of countermeasures and insurance, the cost of mitigation strategies and recovery from cyber-attacks, loss and/or distortion of trade and competitiveness and loss of carried out work (Infosec, 2013). Designing and applying effective strategies against cyber-attacks that accurately transform shared

knowledge into actionable information while maintaining a global view of the network, is imperative in order to confront all of these concerns.

As described in the previous chapter, the current defence mechanisms and monitoring entities nowadays lack capabilities when attempting to take advantage of the knowledge extracted from previous attacks. All these entities and mechanisms have the necessity of effective collection, reporting and sharing data and statistics about previous attacks in order to have readily available and potentially useful information about the state of networks. The utilization of massive analytic capabilities enables a more rapid detection of and reaction to coordinated attacks. This data analytics approach leads to the creation of a general view of networks in their geographical or logical vicinity which allows to detect attacks and understand malicious or suspicious behaviours. This permits to confront attacks at an earlier start and to anticipate future vulnerabilities in order to reinforce the network.

The lack of sharing of information is not the only key issue that hampers the efficient and effective utilization of cybersecurity techniques. Currently, the deployment of specialised hardware-based security appliances is expensive. This deployment generates significant costs (mostly Capital Expenditure - CAPEX) that can be prohibitive for smaller organisations. However, a detect-react strategy that can be applied efficiently across the whole range of the IoT, the end consumer devices, and the backend infrastructure has to be based on mechanisms that allow deploying security functionalities at a fraction of the current costs for today's dedicated hardware.

This approach suggests to move from hardware-based solutions to network virtualisation technologies that permit to build equivalent software solutions instead of hardware solutions. The use of these network virtualisation technologies allows to offer detection and prevention functionalities as services rather than as products. Moving towards a Security-as-a-Service (SecaaS) paradigm allows to provide different types of

security functionalities by using dedicated Service Providers (CSA, 2012). It is evident that a SecaaS solution allows for a more comprehensive view of the entire infrastructure that enables the implementation of threat detection algorithms to a greater part of the network, thus enabling the extraction of greater insight. These algorithms could either be based on traditional, rule-based routines or exploit more unconventional methods, such as machine learning and data mining techniques, which are becoming increasingly prevalent in recent years. Nevertheless, implementing a software solution like cloud computing allows for the utilisation of Big Data analytics as it offers more flexibility in terms of the available resources, while the cost is significantly cheaper compared to hosting such a service in the local infrastructure.

The next subsections provide a brief comparison between traditional and more unconventional threat detection methods while focusing on the theoretical background of the latter.

2.2 Traditional and unconventional approaches to cybersecurity

Cyberinfrastructures are vulnerable due to design and implementation flaws, such as errors in the procedure, code, and design of the software. Malicious users attack system vulnerabilities by using a sequence of events, which helps them to break into a cyberinfrastructure. As a result to these threats, there is a wide spectrum of IDS, varying from antivirus software to hierarchical systems that monitor the traffic of an entire backbone network. The most common classifications are network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). A system that monitors important operating system files is an example of a HIDS, while a system that

analyses incoming network traffic is an example of a NIDS. The differences between these variants are beyond the scope of this thesis.

Another common –and perhaps more interesting– classification of intrusion detection systems is by detection approach. The most well-known variants that are still dominant nowadays are the traditional signature/misuse detection systems. Signature-based IDS refers to the detection of attacks by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware (Lokesak, 2008). This terminology originates from anti-virus software, which refers to these detected patterns as signatures. Although signature-based IDS can easily detect known attacks, it is impossible to detect new attacks, for which no pattern is available. On the other hand, anomaly-based detection systems are detecting deviations from a model of "good" traffic, which often relies on machine learning. They were primarily introduced to detect unknown attacks, in part due to the rapid development of malware. The basic approach is to use machine learning to create a model of trustworthy activity, and then compare new behaviour against this model. Although this approach enables the detection of previously unknown attacks, it may suffer from false positives: previously unknown legitimate activity may also be classified as malicious.

The following subsections §2.2.1 and §2.2.2 will briefly introduce fundamental knowledge, key issues, and challenges in signature/misuse detection systems as well as in anomaly detection systems.

2.2.1 Signature-based detection systems

In essence, signature detection systems, are based on traditional IDS triggering methods that generate alarms when a known cyber misuse occurs. A signature detection technique measures the similarity between input events and the signatures of known intrusions. It

flags behaviour that shares similarities with a predefined pattern of intrusion. Thus, known attacks can be detected immediately and effectively with a lower false-positive rate. Signature detection is used to recognize specifically unique patterns of unauthorized behaviour to predict and detect subsequent similar attempts. These specific patterns, called signatures, include patterns of specific log files or packets that have been identified as a threat. Each file is composed of signatures, which are unique arrangements of zeros and ones. For example, in a host-based intrusion detection system, a signature can be a pattern of system calls. In a network-based IDS, a signature can be a specific pattern of the packet such as packet content signatures or header content signatures that can indicate unauthorized actions such as improper FTP initiation. The packet includes source or destination IP addresses, source or destination TCP/UDP ports, and IP protocols such as UDP, TCP, and ICMP, and data payloads.

Signature detection methods match the learned patterns and signature of attacks to identify malicious users as shown in Figure 2.1. If the learned patterns and signature of attacks match, the system will alert the system administrator that a cyberattack has been detected. Then, the administrator will attempt to label the attack. The related information will be delivered to an administrator. Signature detection methods typically search for known potentially malicious information by scanning cyberinfrastructure and make decisions based on a significant amount of prior knowledge of the attack signatures. For these solutions to work, the security software will need to obtain collections of known cyberattack characteristics. Therefore, the quality and reliability of the signature detection results rely on the frequent updating of the signature database. For example, antispyware tools usually use signature detection techniques to find malicious software embedded in a computational system. When a signature-based antispyware tool is active, it scans files and programs in the system and compares them with the signatures in the database. If there is a match, the tool will alert the system administrator that spyware has been detected and will

provide information associated with the spyware, such as the name of the software, the danger level, and the location of the spyware, to cyber administrators. This technique may often locate a known threat, however it occasionally may cause false alarms. A false alarm is an instance in which an alert occurs although unauthorized access has not been attempted. For example, a user may forget a login password and make multiple attempts to sign into an account. Depending on the robustness and seriousness of a triggered signature, an alarm or notification will be reported to the proper authorities.

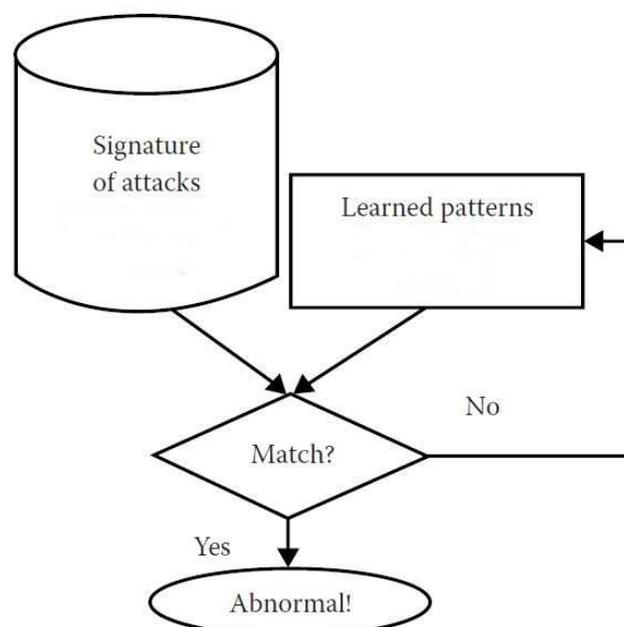


Figure 2.1: Workflow of a typical signature detection system

In general, the strength of a deterministic signature detection system depends on the sufficiency of the existing knowledge of the system vulnerabilities and known attack patterns. That does not mean that cognitive analytics are absent in misuse detection - traditionally though- the construction of the knowledge of a cyberinfrastructure relies heavily on domain experts. Domain experts vary in experience and knowledge, which leads to the incomplete coverage and inaccurate detection of malicious behaviours. Moreover, any variation, evolution, or blending of known attacks can challenge the similarity learning

process. All the above have resulted in the development of more unconventional detection methods that are based on machine-learning techniques, which are described consequently.

2.2.2 Anomaly detection systems

An anomaly detection system will trigger alarms when the detected object has a significantly different behaviour from a predefined set of normal patterns. Hence, anomaly detection techniques are designed to detect patterns that deviate from an expected normal model built for the data. In cybersecurity, anomaly detection includes the detection of malicious activities, such as penetrations or denial of service, and usually consists of two steps: training and detection (Dua et al., 2016). In the training step, machine-learning techniques are applied to generate a profile of normal patterns in the absence of an attack. In the detection step, the input events are labelled as attacks if the event records deviate significantly from the normal profile. Subsequently, anomaly detection can detect previously unknown attacks. However, anomaly detection is hampered by a high rate of false alarms. Moreover, the selection of inappropriate features can hurt the effectiveness of the detection result, which corresponds to the learned patterns. In extreme cases, a malicious user can use anomaly data as normal data to train an anomaly detection system, so that it will recognize malicious patterns as normal.

The goal of anomaly detection is to target any event falling outside of a predefined set of normal behaviours. Anomaly detection programs assume that any intrusive event is a subset of anomalous activity. In this aspect, it is different from misuse detection, which first defines the signature of abnormal behaviour to indicate attacks. Anomaly detection first defines a profile of normal behaviours, which reflects the health and sensitivity of a cyberinfrastructure. Correspondingly, an anomaly behaviour is defined as a pattern in data that does not conform to the expected behaviours, including outliers, abbreviations,

contaminants etc. in applications. When new attacks appear and normal behaviours remain the same, anomaly detection can find the new or unusual attacks and provide an early alarm. Like misuse detection, anomaly detection relies on a clear boundary between normal and anomalous behaviours, where the profile of normal behaviours is defined as different from anomaly events. The profile must fit a set of criteria, for example it must contain robustly characterized normal behaviour, such as a host/IP address or VLAN segment and have the ability to track the normal behaviours of the target environment. Additionally, it should include the following information: occurrence patterns of specific commands in application protocols, association of content types with different fields of application protocols, connectivity patterns between protected servers and the outside world, and rate and burst length distributions for all types of traffic (Gong, 2003). In addition, profiles based on a network must be adaptive and self-learning in complex and challenging network traffic to preserve accuracy. Anomaly detection should detect malicious behaviours including segmentation of binary code in a user password, stealthy reconnaissance attempts, backdoor service on a well-known standard port, natural failures in the network, new buffer overflow attacks, HTTP traffic on a nonstandard port, intentionally stealthy attacks, variants of existing attacks in new environments, and so on. For example, if a user who usually logs in around 9:00AM from university dormitory in Greece, suddenly logs in at 4:30AM from an IP address of Russia, then an anomaly has occurred. It is evident that the accurate detection of these malicious behaviours encounters several challenges. The key challenge is that the huge volume of data with high-dimensional feature space is difficult to manually analyse and monitor. Such analysis and monitoring requires highly efficient computational algorithms in Big Data processing and pattern learning. Furthermore, much of the data is streaming data, which requires online analysis. It is also difficult to define a representative normal region or the boundary between normal and outlying behaviour. The concept of an outlier varies among application domains; the labelled anomalies are not

available for training or validation. Finally, training and testing data might contain unknown noises, as normal and anomaly behaviours constantly evolve.

Despite the aforementioned challenges, however, it should be noted that anomaly detection systems based on machine learning techniques are the sole line of defence against unknown threats, which cannot -by definition- be detected by signature based systems. The most commonly used machine learning methods in anomaly detection systems will be thoroughly described in the next subsection.

2.3 Machine learning methods in anomaly detection systems

This subsection covers the fundamental background knowledge for some of the most popular machine learning methods used in state-of-the-art anomaly detection systems. While the core of any such system is -undeniably- the machine learning technique that is being implemented, it should be noted that all efficient cybersecurity solutions that rely on anomaly detection methods consist of a number of additional components, in order to properly ingest a vast amount of network data, distribute it to computational nodes, perform security analytics and extract mitigation responses. All these components constitute a framework of technologies, tightly connected to the fields of Big Data and distributed computing and will be described in the following chapters.

There exists a plethora of machine learning algorithms; the selection of the most appropriate in each case depends on the nature of the problem, the size and type of the available data and many other factors as can be seen in Figure 2.2. The most prevalent classification of all types of machine learning methods is related to the way they infer a

function from any given data. There are two main categories of machine learning methods; supervised and unsupervised (Mitchell, 1997).

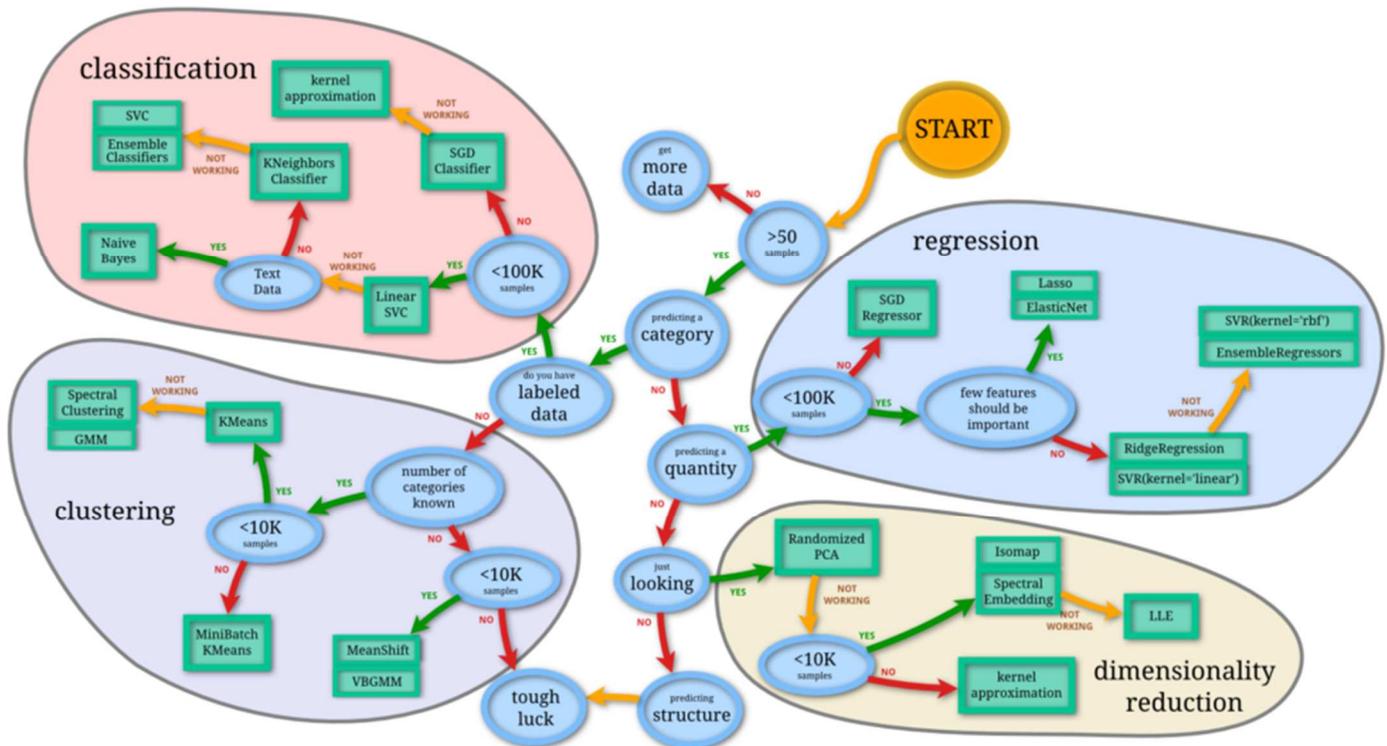


Figure 2.2: Machine learning methods taxonomy

Supervised techniques predict a hidden function using training data. The training data have pairs of input variables and output labels or classes. The output of the method can predict a class label of the input variables. Examples of supervised learning are classification and prediction algorithms. The most popular categorizations used for cybersecurity purposes include artificial neural network, support vector machine, and decision trees.

On the contrary, unsupervised machine learning methods are an attempt to identify hidden patterns without introducing training data (i.e., pairs of input and class labels), meaning that no target or label is given in the sample data. These methods are designed to summarize the key features of the data and to form the natural clusters of input patterns

given a particular cost function. Typical examples of unsupervised methods include k-means clustering, k-nearest neighbour, and self-organization map. Unsupervised learning may be difficult to evaluate, because it does not have an explicit teacher and, thus, does not have labelled data for testing. However, labelled security data necessary for supervised algorithms is sometimes hard to acquire and it will almost always lack some threat classes.

Next follows a brief presentation of the most common supervised and unsupervised machine learning methods followed by a between them comparison. This is expected to provide the necessary background to the reader, in order to fully comprehend the threat detection procedure that will be described in the following chapters.

2.3.1 Supervised machine learning methods in cybersecurity

2.3.1.1 Artificial Neural Networks

The Artificial Neural Network (ANN) is a machine-learning model that transforms inputs into outputs that match targets, through nonlinear information processing in a connected group of artificial neurons, which make up the layers of “hidden” units (Cannady, 1998). The activity of each hidden unit and output \hat{Y} is determined by the composition of its input X and a set of neuron weights W , where W refers to the matrix of weight vectors of hidden layers. When ANN is used as a supervised machine-learning method, efforts are made to determine a set of weights to minimize the classification error. One well-known method that is common to many learning paradigms is the least mean-square convergence. The objective of ANN is to minimize the errors between the ground truth Y and the expected output $\hat{Y} = f(X, W)$ as $E(X) = (f(X, W) - Y)^2$. The behaviour of an ANN depends on both the weights and the transfer function T_f , which are specified

for the connections between neurons. For example, in a typical two-layer ANN shown in Figure 2.3, the net activation at the j^{th} neuron of layer 1 can be presented as:

$$y_j^1 = T_f \cdot (\sum_i x_i \cdot w_{ji}^1) \quad (1)$$

while the transfer function T_f can either be linear, threshold, or sigmoid, each of them providing a different output, depending on the use case.

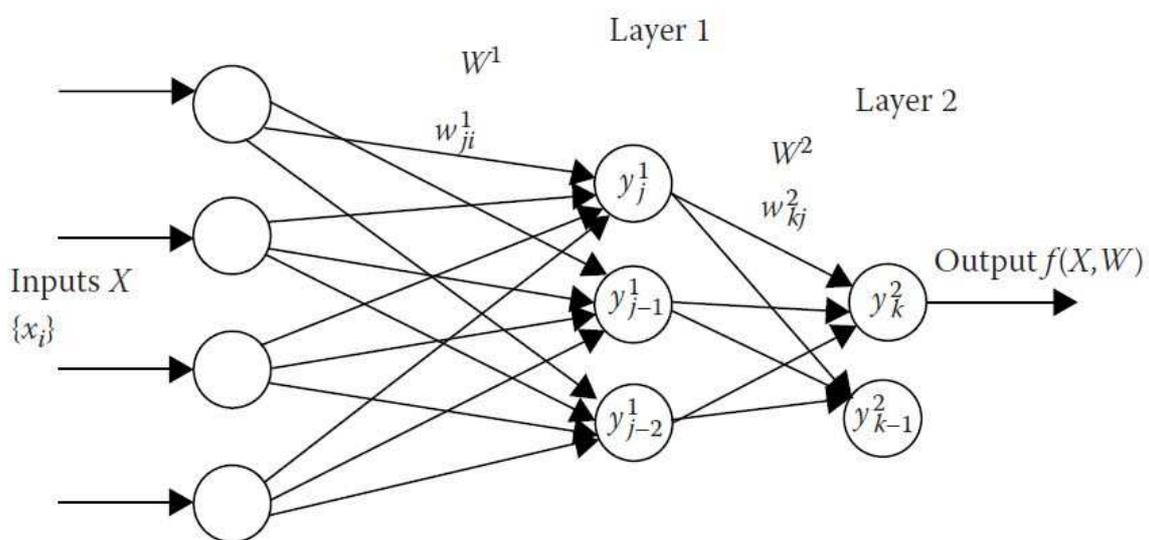


Figure 2.3: Example of a two-layer ANN framework

The most common variant of ANN is the feed forward back-propagation (BP) ANN, where information is transformed from an input layer through hidden layers to an output layer in a straightforward direction without any loop included in the structure of network. In this case, ANN is trained as follows: First, data is being input to the network and the activations for each level of neurons are cascaded forward. The desired output and real output are compared to update BP ANN structure, e.g., weights in different layers, layer-by-layer in a direction of BP from the output layer to the input layer.

Using ANN in anomaly detection systems, aims to provide a generalization from incomplete data and classify online data as a normal anomaly. The back-propagation (BP) ANN feeds input data to the network and compares the desired output and the ground truth, while updating the weights between the different layers. ANN has been applied in cybersecurity by (Ghosh et al., 1999) and (Liu et al., 2002). Each of these studies demonstrates ANN's ability to analyse sequences of system calls, which can then be used to deploy an anomaly detection system. In the first study, system calls were captured using the Sun Microsystem's basic security module (BSM). These data showed regular patterns of behaviour and they were used to build normal software behaviour profiles by capturing the frequencies of system calls. Then, the trained algorithm was used to monitor the behaviour of programs by noting irregularities in their behaviour. The system performed offline intrusion detection in the experiments using the 1998 DARPA intrusion detection evaluation data sets. The experiments showed a 3% false-positive rate (FPR) and 77% of attacks were detected. During the second study, several variants of ANN methods were compared, using two input data encoding techniques (binary and decimal representation). Using binary encoding, the ANNs had lower error rates than decimal encoding. However, decimal encoding appeared to handle noise well, and the classifiers were able to be trained with fewer data.

2.3.1.2 Support Vector Machines

The objective of the SVM algorithm is the classification of data points. Given data points X in an n dimensional feature space, SVM separates these data points with a $n - 1$ dimensional hyperplane, thus classifying them with the hyperplane. Given that there can be an infinite number of hyperplanes that can divide the space in two subspaces, we seek that particular hyperplane that has the maximum distance to the nearest data point on each

side. Such a linear classifier is also called the maximum margin classifier (Hu, 2003). As shown in Figure 2.4, any hyperplane can be written as the set of points X , satisfying $w^T x + b = 0$, where the vector w is a normal vector perpendicular to the hyperplane and b is the offset of the hyperplane from the original point along the direction of w . Given the labels of data points X for two classes: class 1 and class 2, we present the labels as $Y = +1$ and $Y = -1$. Meanwhile, given a pair of (w^T, b) , we classify data X into class 1 or class 2 according to the sign of the function $f(x) = \text{sign}(w^T x + b)$. Thus, the linear separability of the data X in these two classes can be expressed in the combinational equation as $y \cdot (w^T x + b) \geq 1$. In addition, the distance from data point to the separator hyperplane $w^T x + b = 0$ can be computed as $r = (w^T x + b) / \|w\|$, and the data points closest to the hyperplane are called support vectors. The distance between support vectors is called the margin of the separator.

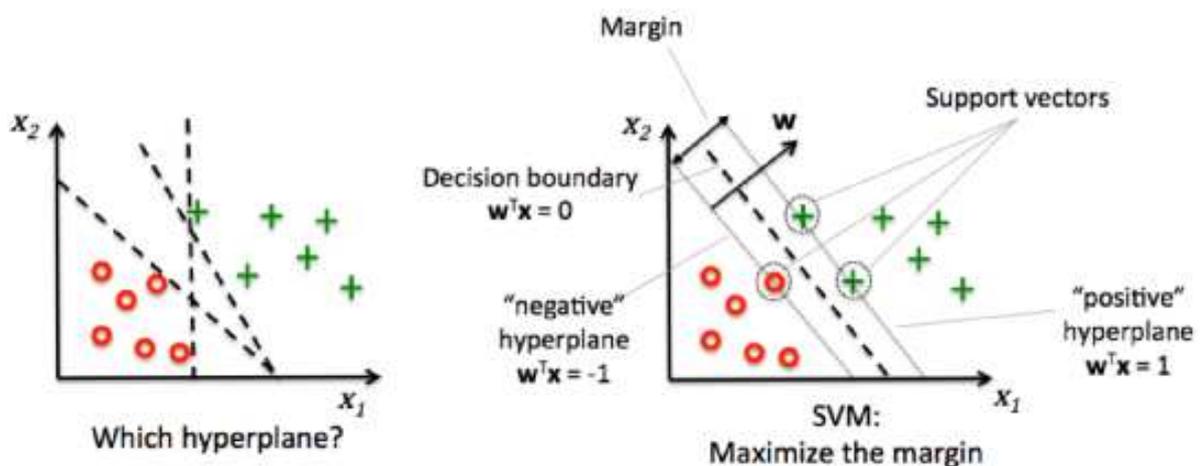


Figure 2.4: Hyperplane and support vectors of a linear SVM algorithm

In linear SVM the hyperplane is calculated by solving the following quadratic optimization problem:

$$\arg_{w,b} \min = \left(\frac{1}{2} \| w \|^2 \right)$$

$$s. t. \quad y \cdot (w^T x + b) \geq 1 \quad (2)$$

In anomaly detection, SVM outperforms ANN because it can achieve the global optimum and easily control the overfitting problem by fine tuning support vectors to separate data. Thus, supervised SVM has been employed by training the SVM structure with both attack data sets and normal data sets by (Chen et al., 2005). They used BSM audit data from the 1998 DARPA intrusion detection evaluation data sets to conduct supervised SVM and compared the results with those obtained using ANN in the same workflow. They collected system call information over processes and extracted frequencies for system calls and processes. They selected the 10 days in which the most attacks appeared in the 7 week training data. Then, they divided the attack data sets into two sets: half for training and half for testing. They implemented SVM classification over the testing data using the two parameters and evaluated the detection result by using a receiver operating characteristic curve (ROC curve), a graphical plot that illustrates the diagnostic ability of a binary classifier system and FAR. In these experiments, SVM outperformed ANN.

2.3.1.3 Decision Trees

A decision tree is a tree-like structural model that has leaves, which represent classifications or decisions, and branches, which represent the conjunctions of features that lead to those classifications. A binary decision tree is shown in Figure 2.5, where C is the root node of the tree, A_i ($i = 1, 2$) are the leaves which represent the terminal nodes of the tree, and B_j ($j = 1, 2, 3, 4$) are branches that represent the decision points of the tree. Tree classification of an input vector is performed by traversing the tree beginning at the root node, and ending at the leaf. Each node of the tree computes an inequality based

on a single input variable, each leaf is assigned to a particular class and each inequality that is used to split the input space is only based on one input variable. A popular variant of decision trees are the linear decision trees which are similar to the binary ones, except that the inequality computed at each node has an arbitrary linear form that may depend on multiple variables. With the different selections of splitting criteria, classification and regression trees and other tree models are developed. A decision tree depends on if-then rules, but requires no parameters and no metrics. This simple and interpretable structure allows decision trees to solve multi-type attribute problems. Decision trees can also manage missing values or noise data. However, they cannot guarantee the optimal accuracy that other machine-learning methods can.

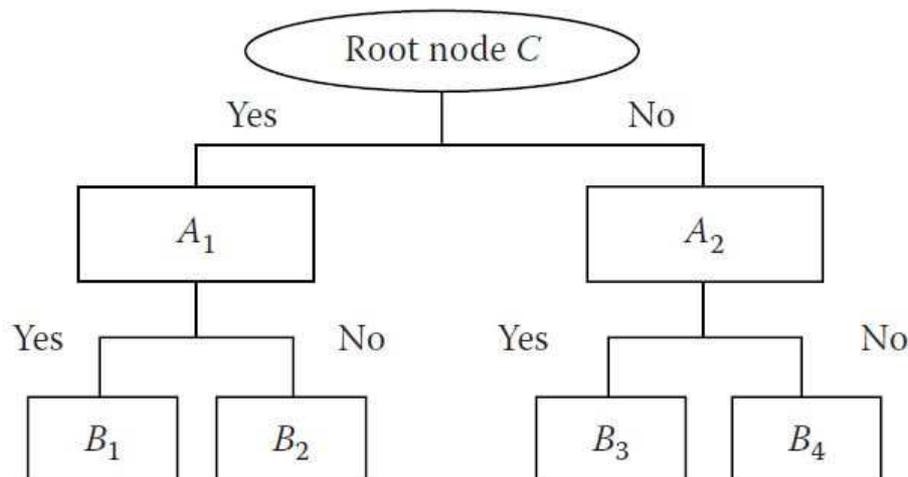


Figure 2.5: Structure of a binary decision tree

Although decision trees are easy to learn and implement, they do not seem to be popular methods of anomaly detection, probably because seeking the smallest decision tree, which is consistent with a set of training examples, is known to be NP-hard (Krueger, 2003). Nevertheless, their use is more common in signature detection and in anomaly classification, the latter being the procedure of matching an anomaly (detected by another machine learning method, like SVM) with a specific known type of threat.

2.3.1.4 Hidden Markov Model

Hidden Markov Models (HMM) are used when the sample datasets are not independent and/or identically distributed. If the provided data is sequential and the samples are correlated, HMM is proposed for solving learning problems of sequential patterns (Rabiner, 1989). In HMM, the observed samples $y_t, t = 1, \dots, T$, have an unobserved state x_t at time t , as shown in Figure 2.6. Each node represents a random variable with the hidden state x_t and observed value y_t at time t . It is assumed that state x_t has a probability distribution over the observed samples y_t and that the sequence of observed samples embed information about the sequence of states. Statistically, HMM is based on the Markov property that the current true state x_t is conditioned only on the value of the hidden variable x_{t-1} but is independent of the past and future states. Similarly, the observation y_t only depends on the hidden state x_t .

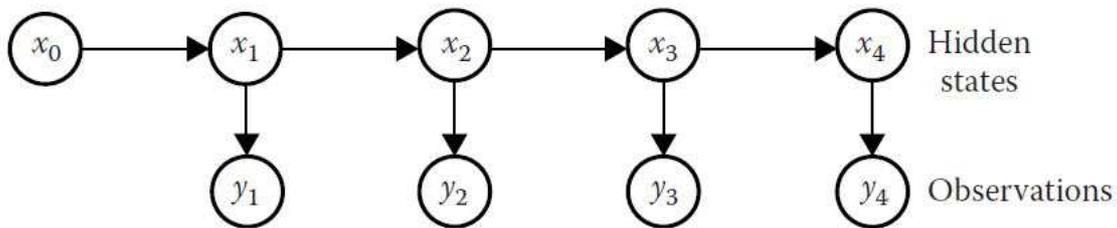


Figure 2.6: Hidden states and observations of a Hidden Markov Model algorithm

A famous solution to HMM is the Baum–Welch algorithm, which derives the maximum likelihood estimate of the parameters of the HMM given a data set of output sequences (Frazzoli, 2013). Using the above information, we begin by considering that $Y = (y_1, \dots, y_T)$ represents the fixed observed samples and $X = (x_1, \dots, x_t)$ the state (both of length T), thus we can have the state set $S = (s_1, \dots, s_M)$ and the observable state $O = (o_1, \dots, o_N)$. We define A as the state transition array, where each element $A_{ij}, (i =$

$1, \dots, M, j = 1, \dots, N$) represents the state transformation from s_i to s_j . The transformation is calculated as:

$$A_{ij} = p(x_t = s_j | x_{t-1} = s_i) \quad (3)$$

We also have to define B as the observation array B_{jk} , ($j = 1, \dots, M, k = 1, \dots, N$) where each element represents the probability of observation o_k having the state s_j :

$$B_{jk} = p(y_t = o_k | x_t = s_j) \quad (4)$$

Finally, we have to define the initial probability array π_t , ($t = 1, \dots, T$) as the probability that the observation y_t has the state s_i , ($i = 1, \dots, \pi$):

$$\pi_i = p(x_i = s_i) \quad (5)$$

After having defined all the above parameters, we can define the Hidden Markov Model as:

$$\lambda = (A, B, \pi) \quad (6)$$

It should be mentioned that, although HMM is an elegant method to classify or predict the hidden state of the observed sequences with a high degree of accuracy when data fit the Markov property, when the true relationship between hidden sequential states does not fit the proposed HMM structure, the model will result in poor classification or prediction. HMM also suffers from large training data sets and complex computation, especially when sequences are long and have many labels.

In anomaly detection, HMMs can effectively model temporal variations in program behaviour (Warrender et al., 1999) (Qiao et al., 2002) (Wang et al., 2006). In order to apply a model a normal activity state set S and a normal observable data set O are needed. Given an observation sequence Y, the objective of HMM is to search for a normal state sequence of X, which has a predicted observation sequence most similar to Y with a probability for

this examination. If this probability is less than a predefined threshold, we declare that this observation indicates an anomaly state. Warrender et al. performed studies on various publicly available system call data sets from nine programs, such as MIT LPR, and UNM LPR. They implemented 40-state HMMs for many of the programs because 40 system calls composed those programs. The states were fully connected and transitions were allowed from any state to any other state. The Baum-Welch algorithm was applied to build the HMM using training data, and the Viterbi algorithm was implemented on the HMM to find the state sequence of system calls. Assuming that in a good HMM, normal sequences of system calls require only likely transitions and outputs, while anomalous sequences have one or more system calls that require unusual transitions and outputs, each system call was tested for tracking unusual transitions and outputs, using a selected threshold for transitions and outputs. The experiments showed that HMM could detect anomaly data quickly and at a lower mismatch rate. It was also concluded that HMM training needs multiple passes through the training data, which takes a great deal of time and -consequently- extensive memory to store transition probabilities during training, especially for long sequences.

2.3.2 Unsupervised machine learning methods in cybersecurity

2.3.2.1 k-Means Clustering

Clustering is arguably the simplest and most popular unsupervised machine learning method. It is defined as the assignment of objects into groups called clusters, so that objects from the same cluster are more similar to each other than objects from different clusters. The similarity of the objects is usually determined by the distance between them over multiple dimensions of the data set. Since clustering is an approach of unsupervised

learning, examples are unlabelled, meaning that they are not pre-classified. A widely used variant of this method is the k-Means clustering, which partitions the given data points X into k clusters, in which each data point is more similar to its cluster centroid than to the other cluster centroids as shown in Figure 2.7 (Murty et al., 1999). The k-means clustering algorithm generally consists of the following steps:

1. Select a number k of initial cluster centroids, $c_1, c_2, \dots, c_{k-1}, c_k$
2. Assign each instance x to the cluster that has a centroid nearest to x .
3. Recalculate each cluster's centroid based on which elements are contained in it.
4. Repeat 2 - 3 until convergence is achieved.

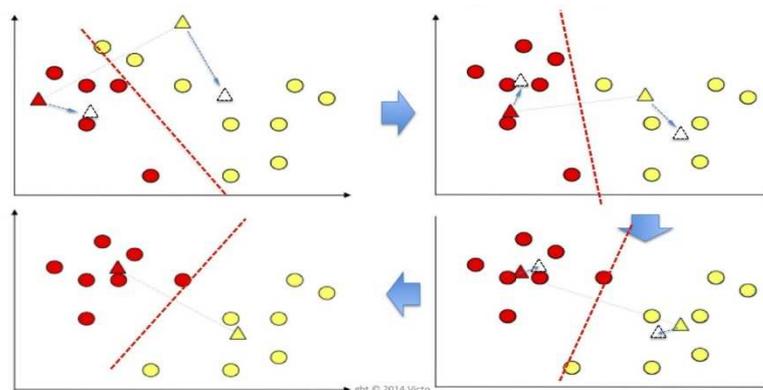


Figure 2.7: k-means clustering example

For the successful implementation of the k-means method, the cluster number k for partitioning and the distance metric must be known, otherwise there is no evaluation method that can guarantee the selected k as well as the distance metric that were selected are optimal. This can be a problem with datasets that we know very little about and fine-tuning is required.

The above clustering anomaly detection method was applied by Portnoy et al. (2001) on the DARPA MIT Knowledge Discovery and Data Mining Cup dataset, which included 4900000 data points with 24 attack types and normal activity in the background. Each data point was a vector of extracted feature values from the connection record obtained between IP addresses during simulated intrusions. The entire dataset was partitioned into 10 subsets

and subsets containing only one type of attacks or full of attacks were removed, so that only 4 subsets were finally left. They performed training and testing several times with different selections of combinational subsets, coming to the conclusion that clustering with unlabelled data resulted in a lower detection rate for attacks than clustering with supervised learning. However, it was acknowledged that unsupervised learning can potentially detect unknown attacks through an automated or semi-automated process, which could not be possible with a trained supervised learning algorithm.

2.3.2.2 Expectation Maximization

The expectation maximization (EM) method is designed to search for the maximum likelihood estimates of the parameters in a probabilistic model. In the case of a histogram of data points, for example, the histogram is regarded as a probability density function whose parameters can be estimated by using the histogram. EM assumes that parametric statistical models, such as the Gaussian mixture model, can describe the distribution of a set of data points and performs two steps iteratively: the expectation (E) step and the maximization (M) step (Dempster et al., 1977). The expectation step computes the log likelihood function of parameters with respect to the current estimate of the distribution for the latent variables. The maximization step uses the above function to compute the parameters that maximize the expected log likelihood found in the expectation step. These parameters are then used to determine the distribution of the latent variables in the next expectation step. Thus, the maximum likelihood function is determined by the marginal probability distribution of the observed data. The EM cycle starts at an initial setting and updates the parameters using a set of equations, converging until its estimated parameters cannot change anymore, an example being shown in Figure 2.8. The EM algorithm can result in a high degree of learning accuracy when the given datasets have the same distribution as the assumption, otherwise the clustering accuracy will be low, since the

model is biased. It can also be used as a fine-tuning method for the parameters of other machine learning algorithms, like LDA, as will be shown later.

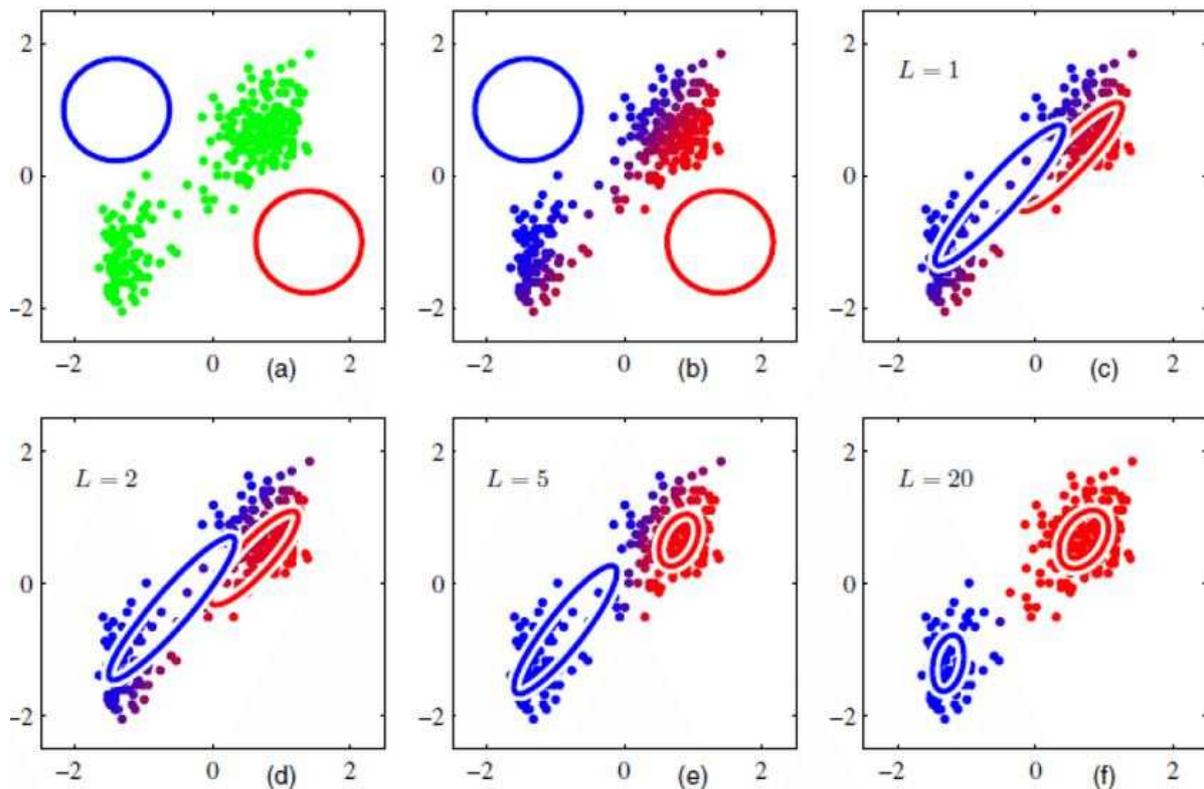


Figure 2.8: Convergence of an EM procedure

2.3.2.3 k-Nearest Neighbour

The k-Nearest Neighbour algorithm (KNN) is a distance-based classification technique that can be found both in supervised and in unsupervised learning. It receives a dataset as input and assigns to each data point the label that has the highest confidence among the k data points nearest to the query point. For example in Figure 2.9, for $k = 5$, the query point X_{query} , is classified to the negative class with a confidence of $3/5$, because there are three negative and two positive points inside the circle. The key components of KNN are the number of nearest neighbours (k) and the distance measure. Generally, a larger k reduces the effect of data noise on classification, but it may blur the distinction between classes,

thus a good practice is that k should be less than the square root of the total number of training patterns. Especially for two-class classification problems, k should be selected among odd numbers to avoid tied votes. As for the distance metric, Euclidean distance is by far the most commonly used, due to its simplicity. The biggest advantages of KNN are its ease of implementation and interpretation and the fact that it does not need to train parameters for learning. On the other hand, using KNN for classification is usually time consuming and storage intensive.

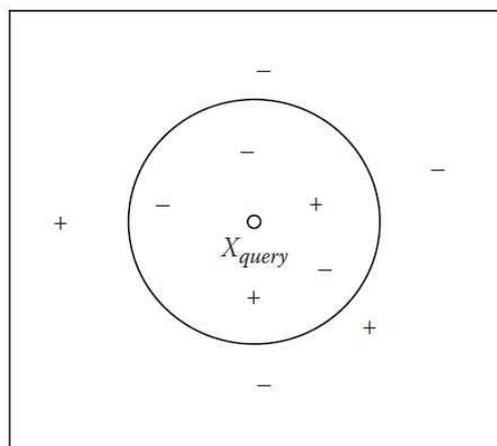


Figure 2.9: k-nearest neighbour example, $k=5$

The most deployed distance-based clustering methods in anomaly detection are adapted from k-means clustering (Liao et al., 2002). If k is not defined in the algorithm, a threshold r is used to constrain the cluster's space (hyper-spheres). Given a data set $X = \{x_1, \dots, x_m\}$ and cluster set $C = \{C_1, \dots, C_N\}$, each data point x_i in X is assigned to the closest cluster C^* , if $dist(x_i, C_j) \leq r$, otherwise the method creates a new cluster C' for this data point, and updates the cluster set. The process is repeated until all data points are assigned to a cluster.

It is apparent that unsupervised KNN has many similarities with k-means clustering, the main difference being the additional constraint r for the clustering threshold. Moreover, as all training data are unlabelled, we cannot determine which clusters belong to normal or

anomaly types, and even so, a cluster may include mixed instances of normal data and different types of attacks. By assuming that normal data greatly outnumbers anomaly data, we label the clusters that constitute more than a predefined percentage of the training data set as normal, while all the others are labelled as anomalies. Meanwhile, threshold r also affects the result of clustering: when it is large, the cluster number will decrease otherwise, the cluster number will increase.

Eskin et al. (2002) deployed a set of algorithms including KNN on the MIT DARPA system call dataset to process unlabelled data by mapping data points to a feature space. Anomalies were detected by finding the data points in the sparse regions of the feature space. First they used fixed-width clustering to compute how many data points were within a fix width of the enquiry data point in the feature space section. Then, they applied KNN by using the sum of distances between the enquiry data and the k nearest neighbours as a score, basing their algorithm on inequalities of triangles. The unsupervised machine learning and detection results were compared against the ground truth, obtaining perfect results on system calls.

2.3.2.4 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation is a generative probabilistic model for collections of discrete data such as text corpora. Being generative means that one doesn't need to provide known class labels in order to infer the patterns. Instead, the algorithm generates a probabilistic model that is used to identify groups of topics (Blei et al., 2003). The probabilistic model is used to classify either existing training cases, or new cases that are provided to the model as input. A generative model can be preferable because it avoids making any strong assumptions about the relationship between the text and categories, and uses only the distribution of words to mathematically model topics. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modelled as a finite mixture over an

underlying set of topics. Each topic is, in turn, modelled as an infinite mixture over an underlying set of topic probabilities. In the context of text modelling, the topic probabilities provide an explicit representation of a document. The goal of the algorithm is to find short descriptions of the members of a collection that enable efficient processing of large collections while preserving the essential statistical relationships that are useful for basic tasks such as classification, novelty detection, summarization, and similarity and relevance judgments.

In order to explain how LDA works, we need to introduce basic notation and terminology. Generally, the language of text collections is used, referring to entities such as words, documents and corpora, mainly to guide intuition, particularly when we introduce latent variables which aim to capture abstract notions such as topics. However, LDA is not necessarily tied to text, and has applications to other problems involving collections of data, including data from domains such as collaborative filtering, content-based image retrieval and bioinformatics and cybersecurity. We define the following terms:

- A word is the basic unit of discrete data, defined to be an item from a vocabulary indexed by $\{1, \dots, V\}$. We represent words using unit-basis vectors that have a single component equal to one and all other components equal to zero. Thus, using superscripts to denote components, the v^{th} word in the vocabulary is represented by a V -vector w , such that $w^v = 1$ and $w^u = 0$ for $u \neq v$.
- A document is a sequence of N words denoted by $w = (w_1, w_2, \dots, w_N)$, where w_n is the n^{th} word in the sequence.
- A corpus is a collection of M documents denoted by $D = \{w_1, w_2, \dots, w_M\}$.

LDA is a generative probabilistic model of a corpus, trying not only to assign high probability to members of the corpus, but also to other similar documents. The basic idea is that documents are represented as random mixtures over latent topics, where each topic

is characterized by a distribution over words. LDA assumes the following generative process for each document w in a corpus D :

1. Choose $N \sim \text{Poisson}(\xi)$.
2. Choose $\theta \sim \text{Dir}(\alpha)$.
3. For each of the N words w_n :

(a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.

(b) Choose a word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n .

The assumptions being made in the above basic model are: (i) the dimensionality k of the Dirichlet distribution and thus the dimensionality of the topic variable z are assumed known and fixed, (ii) the word probabilities are parameterized by a $k \times V$ matrix β , where $\beta_{ij} = p(w^j = 1 | z^i = 1)$ that is to be estimated, (iii) N is independent of all the other data generating variables (θ and z).

Regarding the k -dimensional Dirichlet random variable θ , it can take values in the $(k - 1)$ -simplex and has the following probability density on this simplex:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^k a_i)}{\prod_{i=1}^k \Gamma(a_i)} \theta_1^{\alpha_1-1} \dots \theta_k^{\alpha_k-1} \quad (7)$$

where the α parameter is a k -vector with components $a_i > 0$ and $\Gamma(x)$ is the Gamma function. Given the parameters α and β , the joint distribution of a topic mixture θ , a set of N topics z , and a set of N words w is given by the following equation:

$$p(\theta, z, w | \alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^N p(z_n|\theta)p(w_n|z_n, \beta) \quad (8)$$

where $p(z_n|\theta)$ is θ_i for the unique i such as $z_n^i = 1$. After integration over θ and summing over z , the marginal distribution of a **document** is:

$$p(w|\alpha, \beta) = \int (p(\theta|\alpha) \left(\prod_{n=1}^N \sum_{z_n} p(z_n|\theta) p(w_n|z_n, \beta) \right) d\theta \tag{9}$$

and finally taking the product of the marginal probabilities of single documents, the probability of a **corpus** is:

$$p(D|\alpha, \beta) = \prod_{d=1}^M \int (p(\theta|\alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d) p(w_{dn}|z_{dn}, \beta) \right) d\theta_d \tag{10}$$

The three levels of the LDA representation are made clear by Figure 2.10. The parameters α and β are corpus-level parameters, assumed to be sampled once in the process of generating a corpus. The variables θ_d are document-level variables, sampled once per document. The z_{dn} and w_{dn} are word-level variables and are sampled once for each word in each document.

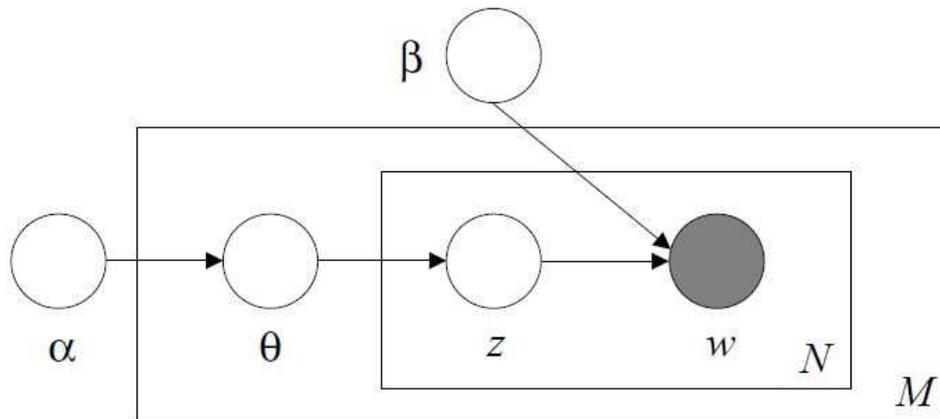


Figure 2.10: Graphical model representation of LDA. The boxes are “plates” representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document.

The parameters α and β are parameters of Dirichlet distributions for (per document) topic and (per topic) word distributions respectively and can be estimated using an empirical Bayes method. Given a corpus of documents $D = \{w_1, w_2, \dots, w_M\}$, we wish to find parameters α and β that maximize the (marginal) log likelihood of the data l :

$$l(\alpha, \beta) = \sum_{d=1}^M \log p(w_d | \alpha, \beta) \quad (11)$$

To achieve this, an alternating variational EM (Expectation-Maximization) procedure is used (see also §2.3.2.2), that maximizes a lower bound with respect to the variational parameters γ and φ , and then, for fixed values of the variational parameters, maximizes the lower bound with respect to the model parameters α and β . It should be noted that each family of lower bounds is indexed by a set of variational parameters, chosen by an optimization procedure that attempts to find the tightest possible lower bound. The iterative algorithm of the EM procedure includes the two following steps:

1. E-step: For each document, find the optimizing values of the variational parameters $\{\gamma_d^*, \varphi_d^* : d \in D\}$. This is performed via finding a tight lower bound on the log likelihood of a variational inference problem, which is translated to an optimization problem, more specifically the one of iteratively minimizing a quantity in order to get the optimal parameter values.
2. M-step: Maximize the resulting lower bound on the log likelihood with respect to the model parameters α and β . This corresponds to finding maximum likelihood estimates with expected sufficient statistics for each document under the approximate posterior which is computed in the E-step.

These two steps are repeated until the lower bound on the log likelihood converges. For symmetric distributions, a high α value means that each document is likely to contain a mixture of most of the topics, and not any single topic specifically. A low α value puts less such constraints on documents and means that it is more likely that a document may

contain mixture of just a few, or even only one, of the topics. Likewise, a high β value means that each topic is likely to contain a mixture of most of the words, and not any word specifically, while a low β value means that a topic may contain a mixture of just a few of the words. In practice, a high α value will lead to documents being more similar in terms of what topics they contain. A high β value will similarly lead to topics being more similar in terms of what words they contain.

Moubayed et al. (2017) have developed a tool that summarises, categorises and models cybersecurity text data, using a text mining approach which models complex inter-relationships between words, documents, and abstract topics. The tool aims at providing security experts with a method to find links, similarities and differences between different documents in a way beyond the current search approaches. The tool is based on the LDA probabilistic topic modelling technique which goes further than the lexical analysis of documents to model the subtle relationships between words, documents, and abstract topics, assisting researchers to query the underlying models latent in the documents and tap into the repository of documents allowing them to be ordered thematically.

Hutchinson's work (Hutchinson, 2014) was the process of transforming collected web-traffic into a term-frequency (T:F) representation form that could allow effective semantic analysis and document classification treatment. For this purpose, LDA was chosen as representative of a modern, Bayesian-model approach for corpus analysis. A collection sensor was deployed on a small network, and collected roughly 24h of web traffic between students and the internet, much of which came from the use of an academic, online learning portal. A tcpdump daemon was used to collect all traffic from a router span-port on the inside network, retaining complete, non-truncated packet contents, and writing the collected traffic to separate files depending upon the IP address of the internal system-browser. LDA processing converged in 12s after 20 overall iterations, after pre-setting the

number of topics to $N=10$. The output from LDA produced an “alpha” vector and a “beta” matrix of size (87326×10) corresponding to the contributions from the words represented in the rows, to the $N = 10$ columns of the beta matrix. A function operated to sort each column in the beta matrix and presented the top 10 words representing each of the 10 topics. The examination of these results showed a roughly equal proportion of English lexicon words and numerical IDs or entities which seems to be a reasonable summary of the web-traffic collected.

Samtani’s study (Samtani et al., 2015) aimed to comprehend the functions and characteristics of assets in hacker forums by applying classification and topic modelling techniques. Using a web crawler routed through the Tor network, the webpages of five popular hacker forums were downloaded. Regular expressions were then used to parse the web pages and store attributes of interest (post, author and thread information) into a MySQL database. After sampling and preparation, LDA was used to understand the topic characteristics of hacker assets. LDA was performed in an identical manner on both the attachment and tutorial postings which were typically descriptive of the type of file and the instructions attached in the posting. Each subset was treated as its own corpus, with each posting being a document. Porter’s stemming algorithm was used to unify words to their common root (i.e., attacking and attack got stemmed to attack) and a stop-words list was used to filter generic terms in the posting. Once stemmed and filtered, 40 topics were extracted from each subset which were manually labelled and tabulated. The results showed that there were several types of attachments available designed to cause direct harm to other machines including the Dirt Jumper DDoS attack, keyloggers and crypters. However, many of the attachments were general, non-threatening resources for other community members to use (books, system security tools, hardware/ software).

Chapter 3

Designing a cybersecurity engine

3.1 Performing anomaly detection in cybersecurity engines

This chapter will address the exploitation of machine learning methods in cybersecurity applications, combining the theoretical background of the two previous chapters. The first two subsections contain a description of the characteristics that comprise a typical anomaly detection workflow and of the mechanism of machine-learning methods as applied in that workflow. The profile and architectural details of the proposed engine will be presented in the following subsections of this chapter.

In general, anomaly detection targets any event falling outside of a predefined set of normal behaviours. Anomaly detection engines assume that any intrusive event is a subset of anomalous activity, thus being different from signature detection systems, which first define the signature of abnormal behaviour to indicate attacks. In anomaly detection, a profile of normal behaviour must first be defined, that reflects the health and sensitivity of a cyberinfrastructure. Similarly, anomalous behaviour is defined as a pattern in data that does not conform to the expected normal behaviour, including mostly outliers, abbreviations or contaminants. When new attacks appear and normal behaviour remains the same, anomaly detection can find the new or unusual attacks and trigger an alarm. For this reason, an anomaly detection system requires as clear boundary as possible between normal and anomalous behaviour, namely clearly defined normal and abnormal profiles. These profiles must be based on a set of criteria such as IP addresses, specific network ports, protocols, content types, rate and burst length distributions for all types of traffic,

size of transmitted packets etc. (Gong, 2003). In addition, profiles based on a network must be adaptive and self-learning in complex and challenging network traffic to preserve accuracy and a low false alarm ratio. Anomaly detection should detect malicious behaviours including reconnaissance attempts, backdoor services on a standard port, network attacks, buffer overflow attacks, HTTP traffic on a nonstandard port, intentionally stealthy etc.

The relevant data that is being input to anomaly detection cybersecurity engines is defined by huge volume and high-dimensional feature space, making it impossible to manually analyse and monitor. Such analysis and monitoring requires highly efficient computational algorithms in Big Data processing and pattern learning as well as proper data ingestion and transformation frameworks. The engine usually tries to take advantage of the fact that normal behaviour is far more usual than the abnormal, setting a representative boundary between the two, like a probability threshold. However, while the same malicious data may occur and be successfully detected, other rarer variations of similar malicious data may lead to an imbalanced data distribution of normal and anomaly data and induce a high false alarm ratio. Moreover, the concept of an anomaly or outlier is different among application domains requiring different information to address different threats. Usually, there are no labelled anomalies available for the training of supervised machine learning algorithms, or even if they are available they might contain noise. Supervised machine learning methods also need attack-free training data, however this kind of training data is difficult to obtain and/or expensive in real-world network environments. Under the optimal scenario, supervised learning algorithms are most probably not efficient enough against the “unknown unknowns” threats that were discussed in the previous chapter, leading to the utilisation of unsupervised learning methods that actually have a chance against constantly evolving malicious behaviours. Unsupervised anomaly detection can overcome the drawbacks of supervised anomaly detection. Thus, semi-supervised and unsupervised machine-learning methods that group the normal patterns by following

similarity measures between the patterns and list the outliers in the abnormal candidate pool are frequently employed (Eskin et al., 2002). Anomaly detection has the ability to detect novel attacks, however it may trigger high rates of false alarm, as these methods flag any significant deviation from the baseline as an intrusion. Thus, it is likely that non-intrusive behaviour falls outside the normal pool and be labelled as an intrusion, resulting in a false positive.

The following subsections will explore the common characteristics that are met in most modern cybersecurity engines and will conclude to the presentation of the one that was developed during the SHIELD project, focusing on the data analytics part.

3.2 Profile of an anomaly detection engine

Based on modern bibliography (Dua et al., 2016), a typical anomaly detection system consists of five discreet steps: data collection, data pre-processing, normal behaviour learning, anomaly detection, and security response or remediation as shown in Figure 3.1 and described below:

- **Data collection:** In data collection, a set of system processes is responsible for the capture of traffic that will later be processed by the engine. Due to the extreme volume of the collected data, it may require reduction.
- **Data pre-processing:** The data pre-processing step includes processes for feature selection, feature extraction, or dimensionality reduction that will allow for a more manageable dataset.
- **Normal behaviour learning:** When a collected dataset is properly transformed, machine learning methods are used to build normal profiles that will help identify malicious outliers. That can be extremely difficult to be done in an efficient way, as hackers often modify malicious codes or data to make them similar to normal patterns. When such an attack occurs, the intrusive behaviour has a high probability of being established as part of the normal profile. When an attack is

missed because it is judged to be part of the normal profile, a false negative occurs.

- **Anomaly detection:** This step is tightly bound to the above, making the practical discretisation between the two steps rather difficult, as the machine-learning algorithm will usually perform the outlier identification procedure right after the normal profiles have been built. In other words, anything that does not belong to the normal profiles is likely to be considered an outlier/anomaly. An additional step that correlates an anomaly to a specific type of threat (if possible) is usually performed here.
- **Security response:** When the anomaly detection phase is concluded, cybersecurity engines can provide a variety of responses depending on their remediation capabilities, the type of the detected anomaly and the user needs. Modern implementations include features varying from basic reports of anomalous activity using a web interface (e.g. dashboard) to more complex operational analytics tools that allow for manual or (semi)automated responses like IP blocking, sandboxing or honeypots.

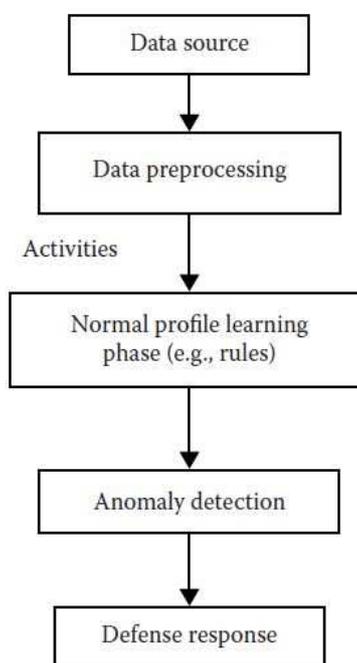


Figure 3.1: Workflow of an anomaly detection system

3.3 Overview of the DARE, a modern IDPS based on machine-learning

The **Data Analysis and Remediation Engine (DARE)** is an information-driven IDPS platform developed as part of the SHIELD project (Figure 3.2) that encompasses a number of subsystems in order to provide cybersecurity and threat mitigation services to large networks. It stores and analyses heterogeneous network information (network traffic), previously collected via monitoring virtual Network Security Functions (vNSFs) such as firewalls, rule-based IDSs etc. It features cognitive and analytical components capable of predicting specific vulnerabilities and attacks. The processing and analysis of large amounts of data is carried out by using Big Data, data analytics and machine learning techniques.

By processing data and logs from vNSFs deployed at specific strategic locations of the network, the DARE components provide feedback to cybersecurity data topologies and - in case malicious activity is detected- they implement remediation activities as a response, either by recommending actions by means of a dashboard and accessible API, or by (optionally) triggering task-specific countermeasures. The DARE platform provides flexible support for both new security capabilities and reconfiguration of existing security controls and allows extensions with multiple data analytics engines by providing a clear API to work with the collected data. Next follows a brief description of the DARE platform.

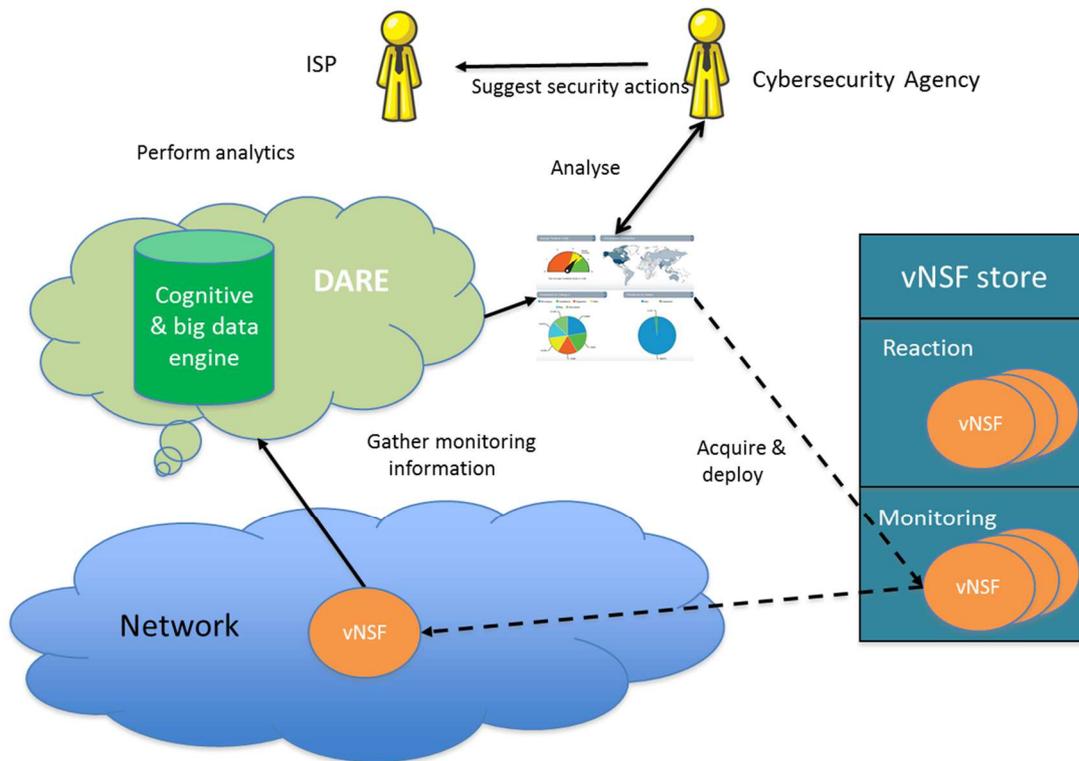


Figure 3.2: Overview of the SHIELD platform

The engine materializes the 5 discreet steps of modern cybersecurity engines as described in the above subsection. It consists of three main components, the **Data Acquisition and Preparation module**, the **Data Analytics Engine** and the **Remediation Engine**, which are briefly showcased below and explained thoroughly in the next subsections.

The **Data Acquisition and Preparation module** is responsible for the ingestion of the selected traffic datasets and their preparation for further processing. This module is composed by three workers, each of them responsible for accumulating different kind of network traffic (flow, DNS, proxy):

- a worker to collect data about network flows (.nfcapd files)
- a worker to collect information regarding DNS (.pcap files)
- a worker to collect web proxy information (bluecoat logs)

These three aspects are considered the main data sources that can be used by any IDPS. More information including examples of their format and content will be given in the next chapter, along with the technical specifications of the analytics module. At the data preparation stage, a number of steps is performed in order to ensure the validity of the ingested data: i) cleaning to remove erroneous samples; ii) curating by adding metadata that helps in the indexing process; iii) enriching the samples by correcting misspellings or missing fields; and iv) integrating datasets if necessary.

The **Data Analytics Engine** leverages two different Data Analytics modules (while opening the platform for the inclusion of others in the future) that use a wide range of complementary detection techniques along with open source frameworks and solutions:

- The **Cognitive Data Analytics module** is able to produce packet and flow analytics, and utilize scalable machine-learning techniques. To this end, it involves the state-of-the-art distributed-computing technologies to allow for batch and streaming processing of large amounts of data, scalability and load balancing, open data models and concurrent running of multiple machine-learning applications on a single, shared, enriched data set. The threat detection procedure of the cognitive module consists of 3 steps for its overall threat detection process and are shortly referred to as *ingestion*, *machine learning* and *operational analytics*. First, a parallel ingest framework is used to decode flow and packet data, then loading the data and its structures in a distributed file system (HDFS). The decoded data are stored in multiple formats so in order to be available for searching, being used by machine learning, being transferred to law enforcement, or being inputted to other systems. Subsequently, the system uses a combination of machine learning tools to run scalable machine learning algorithms (e.g. LDA), not only as a filter for separating bad traffic from begin, but also as a way to characterize the unique behaviour of network traffic. Finally, and in addition to machine learning, a process of context enrichment, noise filtering, whitelisting, and heuristics is applied to network data, in order to present the most likely patterns that may comprise security threats. The above three discrete procedures can be

configured as separate computational nodes (physical or virtual machines) or as a part of a larger distributed computing system.

- A dependable **Security Data analysis module** based on proprietary software, that is mostly focused on rule-based signature detection and pattern matching, including algorithms for the detection of malicious network behaviour, which is adapted and adjusted to the DARE's requirements. The module implements techniques (signature-based, anomaly-based and/or stateful protocol analysis detection) to allow the processing of a wide range of security data sets (e.g. DNS, networking information, web proxy, IP-MAC address mappings, etc.) collected via the vNSF modules. Algorithms include data aggregation, analysis, correlation and detection of unusual networking traffic, domain names, correlations; anomaly detection techniques based on current and historical data. The above module is adapted to the DARE in order to collaborate with the cognitive Data Analysis module, covering different techniques and approaches that improve the analysis results done by SHIELD, however it will be considered as a "black box" for the scope of this thesis. The main focus will be given to the cognitive Data analytics module as an autonomous cybersecurity engine based on machine learning.

Finally, the **Remediation engine** uses the analysis done in the data analytics modules and is fed with alerts and contextual information to determine a mitigation plan for the existing threats. Its main goal is to incorporate a combination of recommendations and alerts that provide relevant threat details to all interested parties using the dashboard and the direct application of countermeasure activities by triggering specific vNSFs. Available information generated by the engine can be used in order to assist Internet Service Providers (ISP) and Computer Emergency Response Teams (CERT) management decision-making.

3.4 Architecture of the DARE

The Data Analysis and Remediation Engine (DARE) is one of the three central innovation pillars of SHIELD, together with the vNSF ecosystem and the hardware attestation (which will not be explained, since they are beyond the context of this thesis). The DARE centralises the management information of SHIELD and exchanges information with all the other components of the solution, as described below (Figure 3.3):

- the **vNSFs**, since the DARE centralises the information obtained from the monitoring vNSFs
- the **vNSF Orchestrator**, since the DARE needs information of the network per tenant (e.g. ISP, ISP clients using SecaaS or a Cybersecurity agency) in order to provide accurate and complete recommendations;
- a **Trust Monitor**, since the DARE needs to know if a vNSF, or even a complete node, has been compromised;
- the **Dashboard**, since the DARE notifies to the dashboard the detected network anomalies and one or more recommendations of network services (set of vNSFs) with their appropriate high-level policies for configuration;
- the **Store**, since the DARE needs to know the vNSFs and NS for deployment or reconfiguration.

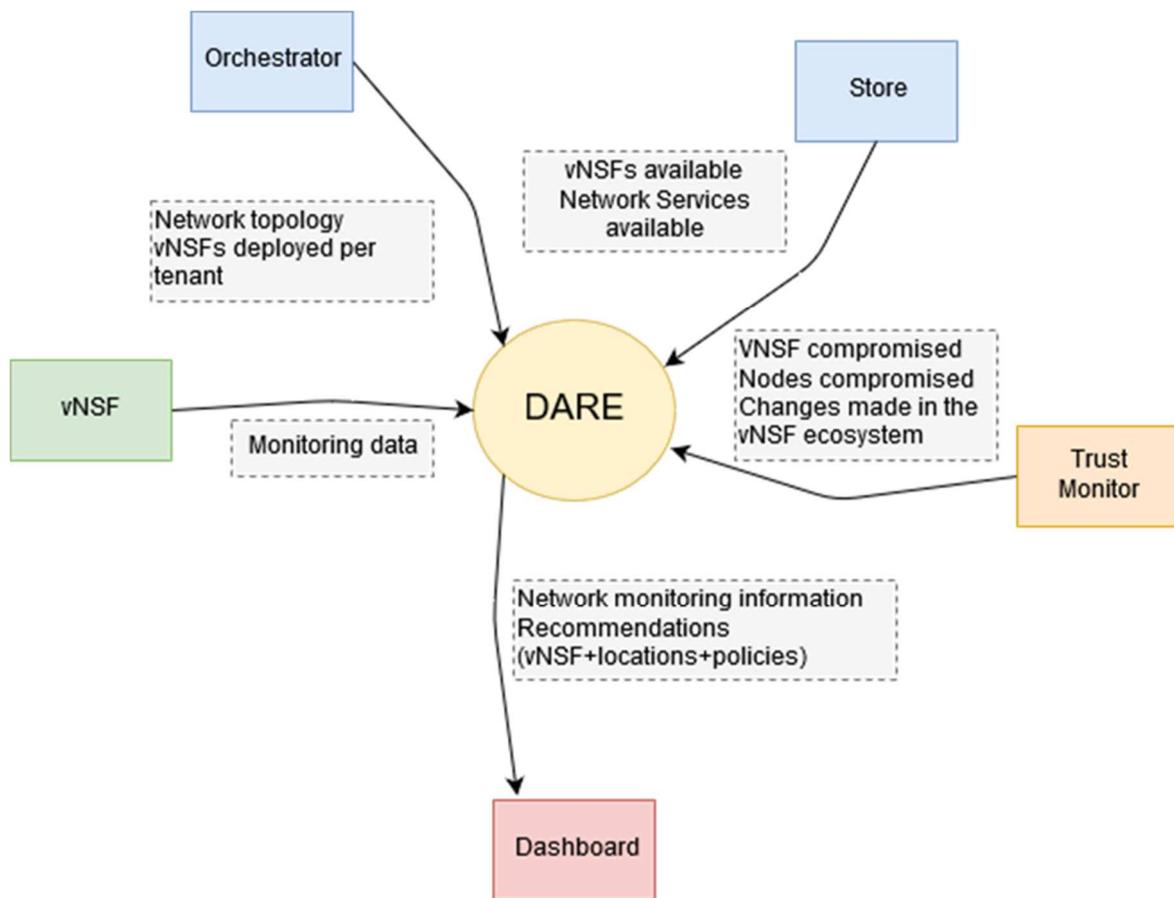


Figure 3.3: Dataflow diagram between the DARE and other SHIELD components

The DARE is composed by a central data analytics engine and a distributed set of data collection components. It is worth mentioning that it has been designed following a Big Data approach where the data value elicitation is divided into three different phases, as shown in Figure 3.4:

- Data acquisition and storage phase
- Data analysis phase
- Cybersecurity topologies phase

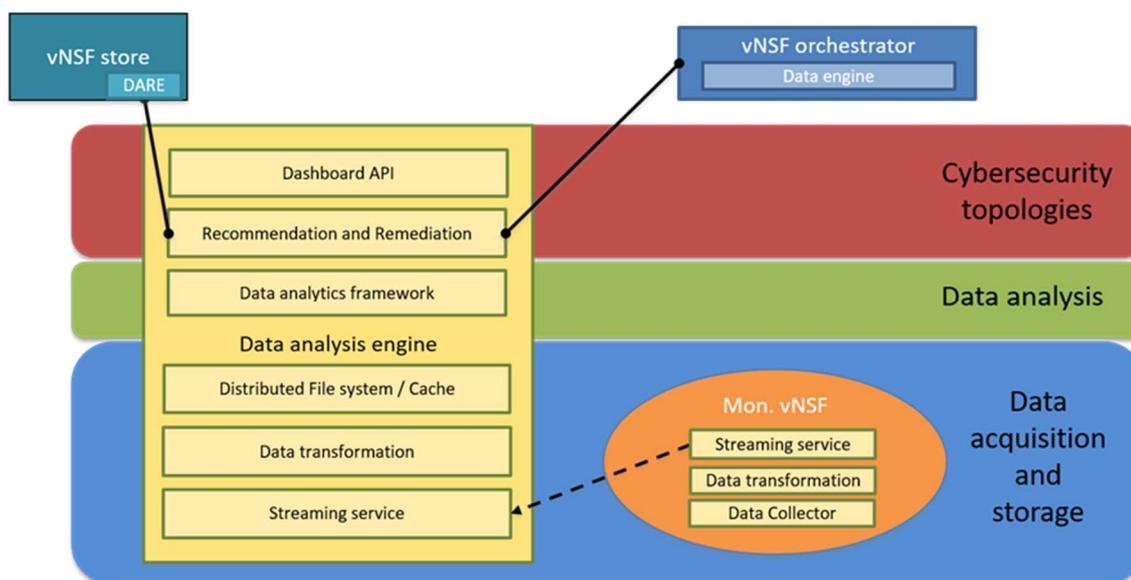


Figure 3.4: The three phases of the DARE platform

Each phase has a number of functionalities that are explained below:

- **Data Collector:** it is responsible for acquiring the data generated in a monitoring vNSF, using the specific format of the technology provided by such vNSF. The collector is part of each monitoring vNSF and integrated into the diagram for clarification purposes.
- **Data transformation:** it is responsible for transforming the format-specific data into a generic format.
- **Streaming service:** it will send the information from the monitoring vNSF to the data analytics central engine, assuring reliability on the communication.
- **Distributed File System/Cache:** it will be responsible for storing the collected data for both, batch (i.e. hard disks) or real-time (i.e. cache) processing.
- **Data analytics framework:** it is responsible for classifying the traffic for anomaly detection using machine learning techniques.
- **Recommendation and remediation:** it will propose, given a specific anomaly or threat detected, a set of vNSFs with the appropriate policies to be deployed in specific places of the network.
- **Dashboard API:** it will push all the generated information to the Dashboard.

A detailed description of the three main DARE phases, the **Data Acquisition and Preparation phase**, the **Data Analytics phase** and the **Remediation phase** follows in the next subsections.

3.4.1 The Data Acquisition and Preparation phase

The Data Acquisition and Preparation phase is responsible for the efficient and reliable capture and storage of various heterogeneous data. It involves mechanisms and methods in order to capture and transfer files generated by network tools to the central data analytics engine. This phase is of high importance for ensuring the integrity of the data and their quality in further processing steps.

Heterogeneous network information is captured via specialised vNSFs, which collect overall networking events that are relevant to threat detection. This information is transferred to the central data analytics engine, where it is stored for further processing. This phase will gather, transform and store the acquired network data to a format that can be processed by analytics components.

DARE leverages two options for describing the low-level architecture of the Data Acquisition and Storage phase:

- Option 1 – Centralised architecture (Figure 3.5): only the collection of the data is distributed, while all the other functionalities are centralised in the Data analysis phase.
- Option 2 – Distributed architecture (Figure 3.6): the data collection and the data transformation are distributed per vNSF and hence, the data is sent to the central engine in a standard format (e.g. CSV).

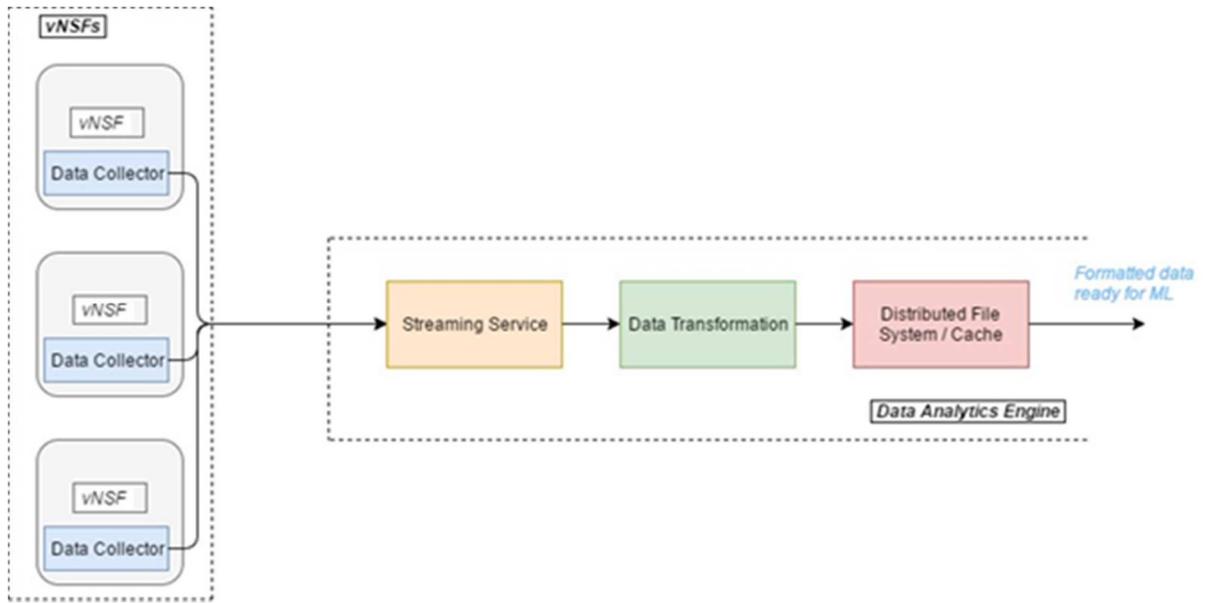


Figure 3.5: Centralised low-level ingestion architecture

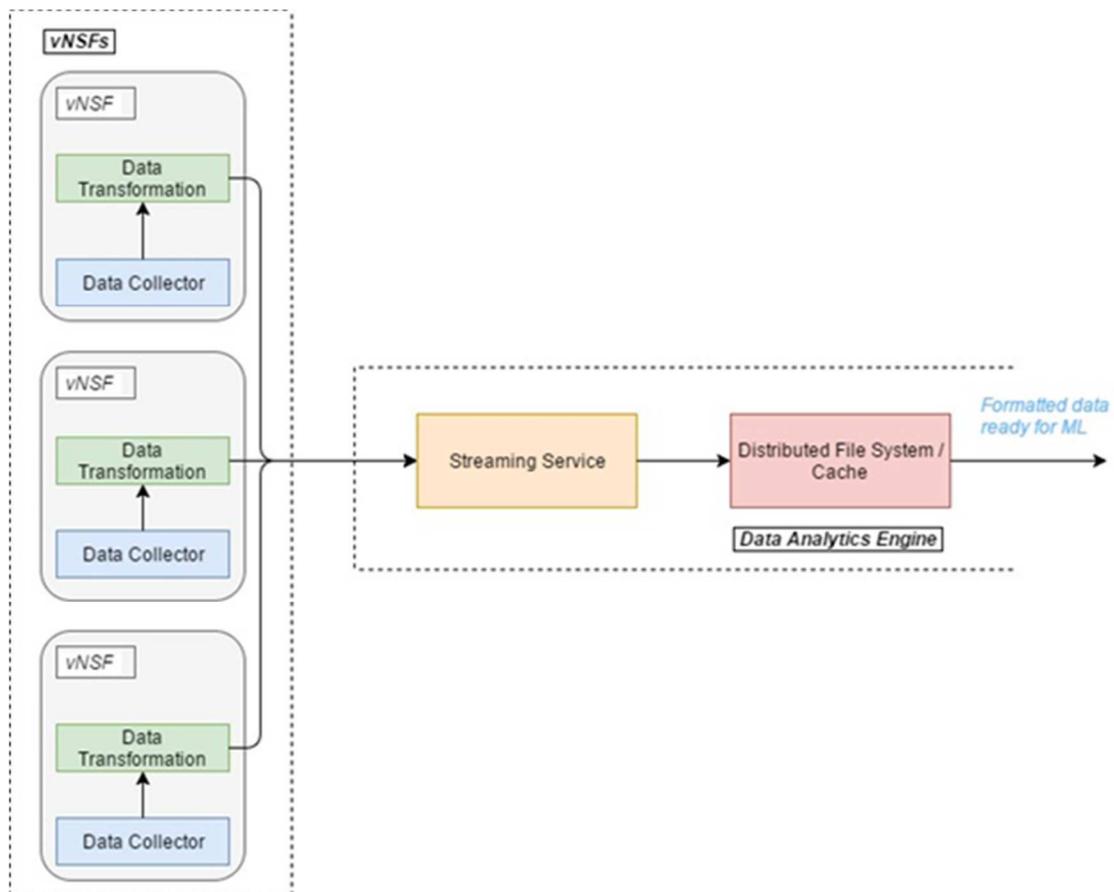


Figure 3.6: Distributed low-level ingestion architecture.

The objective of the data acquisition phase is to gather all the network information produced by the vNSFs, transform it into a generic format, ingest it into the data analytics central engine, and store it for further processing. The low-level architecture of the Data Acquisition and Preparation phase is divided in four main subcomponents.

- **Data Collector:** The data collector subcomponent is the only one which is distributed (one collector per vNSF) in both options. Each vNSF uses a daemon, called data collector, which is responsible for monitoring the vNSF and detecting new files produced by it.
- **Streaming Service:** The acquisition of network data is achieved via a distributed streaming service that splits the network data into smaller specific topics and smaller partitions, while creating a data pipeline for each topic. It has to be reliable and fault tolerant for ensuring the integrity of the data and their quality in further processing steps.
- **Data Transformation:** This is the subcomponent that determines the chosen option for the low-level architecture. If the data transformation is centralised, the architecture is considered to address option 1, while if it is distributed, the architecture is considered to address option 2. In the case of option 1, each pipeline created by the streaming service transfers the stored data to specific daemons which exist inside the central data analytics engine. These daemons are subscribed to a specific topic and partition of the streaming service, and transform the raw network data into a human-readable format, by using dissection tools. They are tasked with reading, parsing and storing the data in a specific distributed format to be consumed by the machine learning algorithms. In the case of option 2, each network data file that is captured by the Data Collector, is sent to specific daemons which exist inside each vNSF (distributed). These daemons transform the raw network data into a human-readable format, by using dissection tools.
- **Distributed File System/Cache:** Once the network data has been transformed, the input is stored in a distributed file system in both the original and modified formats (in the case of option 1) or only the modified (in the case of option 2). The distributed file system is responsible for storing the collected data and making them available, so that it can be accessible by search queries.

3.4.2 The Data Analytics Phase

This phase is composed only by one subcomponent, the data analytics framework (Figure 3.7). This subcomponent features cognitive and analytical functionalities capable of detecting network anomalies that are associated with specific vulnerabilities or threats, offering batch and streaming incident detection. The processing and analysis of large amounts of data is carried out by using Big Data analytics and machine learning techniques. By processing data and logs from vNSFs deployed at specific strategic locations of the network, the data analytics framework is able to link traffic logs that are part of a specific activity in the network and detect any possible anomaly. In case malicious activity is detected, it informs the remediation and recommendation engine.

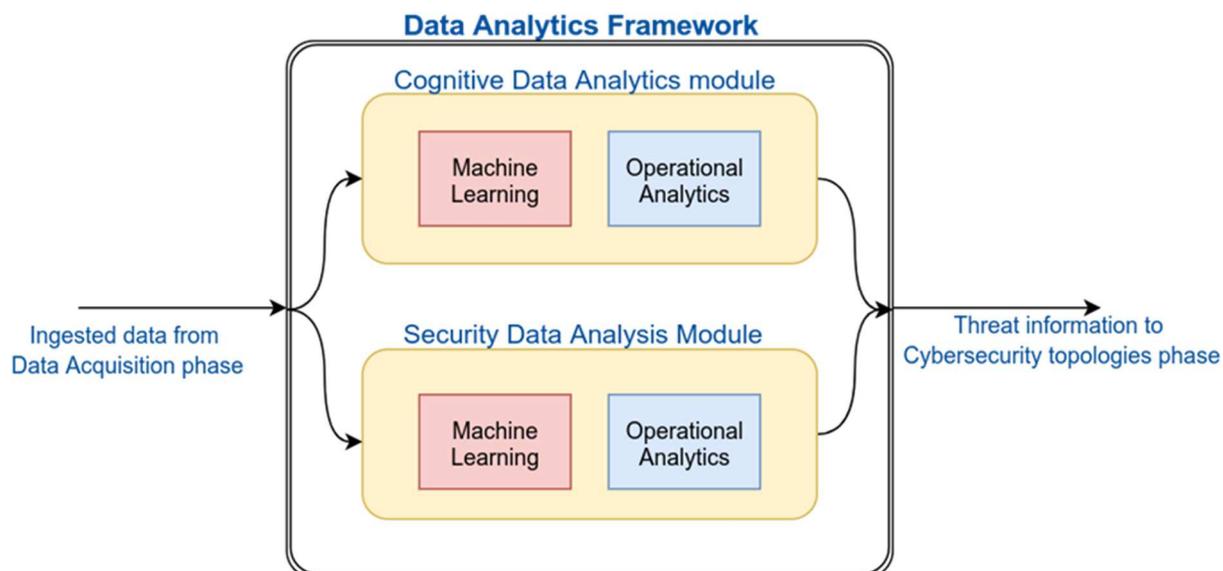


Figure 3.7: Data Analytics Framework overview

The Data Analysis Engine leverages two different data analytics modules that use a wide range of complementary detection techniques along with open source frameworks and solutions, a **Cognitive Data Analytics module** based on open-source technologies

and a proprietary **Security Data Analysis module**. The individual functionalities of each module are described in detail below.

Since the network data provided by the Data Acquisition phase is required for the functionality of both the Cognitive Data Analytics module and the proprietary Security Data Analysis module in the same distributed format, the accessing method is shared between these two modules.

Below we describe the two main Data Analytics modules that comprise the DARE, namely the **Cognitive Data Analytics module** and the **Security Data Analysis module** as well as the anomaly detection procedure that is performed by using them.

3.4.2.1 Cognitive Data Analytics module

This module is an entity of elements that is able to produce packet and flow analytics by using scalable machine-learning techniques. It consists of two main entities that comprise the overall detection procedure, shortly referred to as machine learning and operational analytics and are configured either as separate computational nodes or as a part of a larger distributed computing system. A description of each entity is given below:

- **Machine learning:** The machine learning entity (Figure 3.8) is responsible for the detection of anomalies in network traffic that will lead to the prevention or mitigation of potential threats. For this purpose, DARE uses a combination of open-source tools to run scalable machine-learning algorithms. The machine-learning entity works not only as a filter for separating bad traffic from benign, but also as a way to characterise the unique behaviour of network traffic. It contains routines for performing suspicious connections analytics on flow, DNS, proxy logs, security event data and metrics gathered from the Data Acquisition phase and the built-in Distributed storage system subcomponent. These analytics consume a collection of network events in order to produce a list of the events that are considered to be the least probable, and these are considered the most suspicious. The type of analytics that are utilised by the

machine learning component inside a cluster computing framework, are anomaly Detection algorithms, which provide the statistical model being used for discovering incongruences or rare behaviours by examining network traffic and additional algorithms that aim to enhance the overall detection capabilities of the platform as well as allow the correlation between the detected anomalies and specific threats by classifying the results of the anomaly detection algorithms.

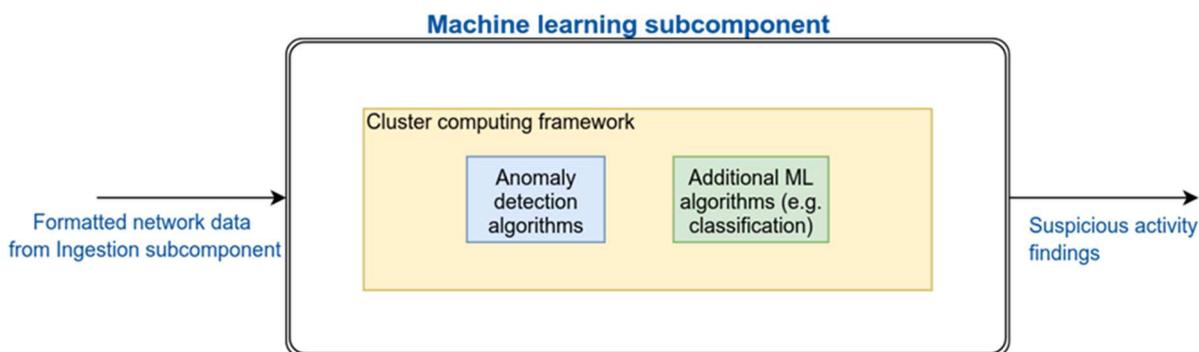


Figure 3.8: Machine learning entity overview

- **Operational analytics:** The OA entity (Figure 3.9) consists of a number of context enrichment, noise filtering, whitelisting and heuristics processes to produce a list of the most likely patterns which may comprise security threats. It provides utilities to extract and transform data, by loading the results into output files. It supports basic data types such as flow, DNS and proxy logs that correspond to the most common types of network threats. The output of the OA entity can be used in the Remediation Engine, to provide recommendations to the users or to optionally activate task-specific countermeasures in the form of security functions from the vNSF Store. It also offers a combined view of the above information in the form of information to be pushed to the Dashboard for better visualisation purposes. A description of each element is given below.
 1. Threat interaction tools: An interactive tool that allows for a comprehensive interaction with the network anomalies detected by the machine-learning component.

2. Ingest summary: It presents the amount of network data that has been ingested on the cluster.
3. Filtering tools: A set of tools that provide the ability for customised results based on time, source/destination, severity, type etc.
4. Whitelisting tools: A convenient set of tools to exclude some of the detected anomalies from the results, thus dealing with potential false-positives.

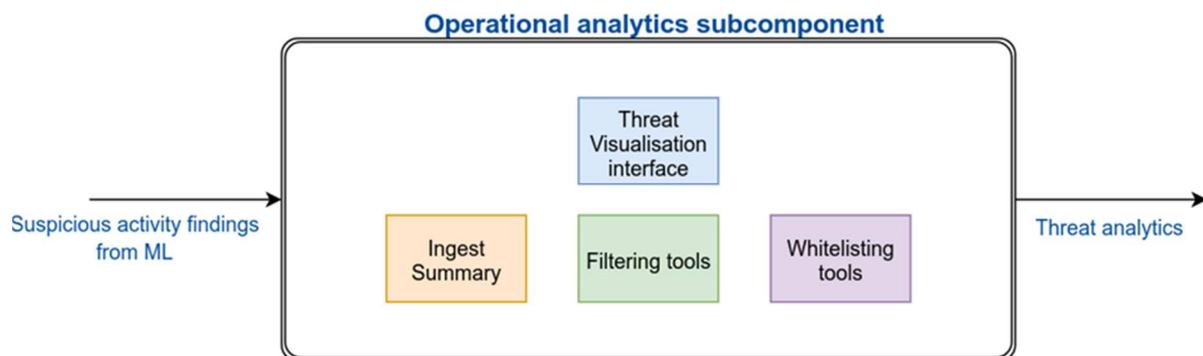


Figure 3.9: Operational analytics subcomponent overview

3.4.2.2 Security Data Analysis module

The Security Data Analysis module (Figure 3.10) is an entity based on a combination of Big Data analytics and machine learning techniques that can efficiently process and analyse a vast amount of network data online and automatically discover and classify cybersecurity threats. This engine follows the architecture designed by the commercial network anomaly detector commercialised by Talaia Networks, which has received very good feedback from customers around the world because of its comprehensive detection of network attacks and its low false positive rate. The Security Data Analysis module has four subcomponents that are described in more detail later. Briefly, this module receives the network data from the distributed storage system /cache subcomponent. The main input used is flow aggregated data (e.g., netflow), however, it also can utilise other sources of information (e.g., DNS, network logs). This data is provided to the Event Clustering and

the Service Profiling subcomponents that compute several measurements and statistics that are used in the Alarm Correction subcomponent to detect anomalous behaviours related to security issues. Once an anomalous behaviour is detected the Anomaly Classifier subcomponent is in charge of classifying it among different network attacks (e.g., DDoS, network scan). All this information is then outputted from the Security Data Analysis to the Remediation module.

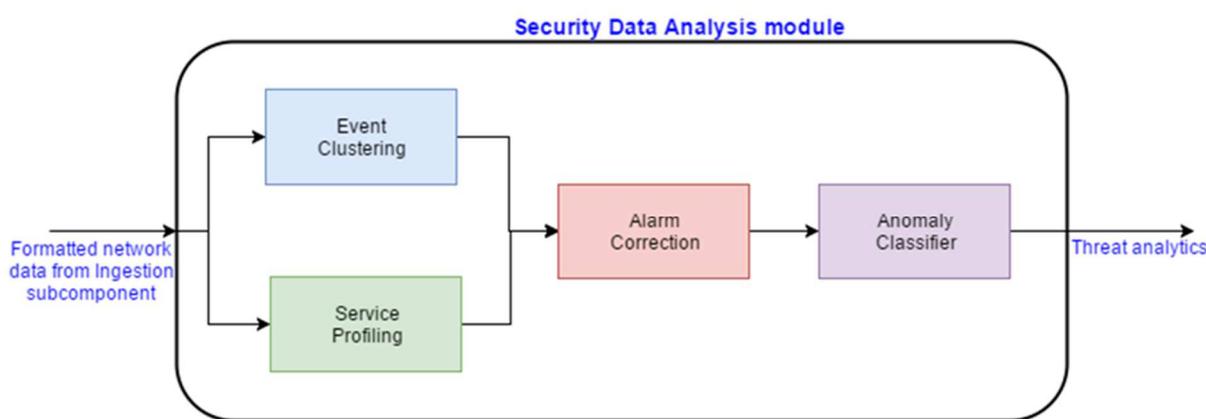


Figure 3.10: Security Data Analysis module overview.

Next follows a brief description of the module's functionalities:

- 1) **Event Clustering:** The Event Clustering entity consists of the adaptation of data mining techniques able to discover multi-dimensional patterns of network usage in the data provided. This entity detects clusters of events in the data which are frequent and share a specific behaviour. The detected clusters, although they can be related to benign traffic, are suspicious of being anomalous traffic that can be related to security issues. The data involved in the cluster and the specific statistics and measurements shared by the events of the cluster are sent to the Alarm Correction in order to decide if the detected cluster is actually related to a security issue.
- 2) **Service Profiling:** The Service Profiling entity performs a thorough and detailed analysis of the data provided in order to create profiles of all the services contained in such data. To perform this analysis in real-time, the engine is

relying on extremely efficient data structures combined with cutting-edge data mining techniques. The objective of this entity is to identify and understand the behaviour of the different services in the data. In addition, this entity is also able to discover the services that are prone to be affected by networks attacks. Similarly to the Event Clustering entity, the information related to the services identified is then forwarded to the Alarm Correction entity for further analysis.

- 3) Alarm Correction: The Alarm Correction entity receives as input the output from the Event Clustering and the Service Profiling entities. By comparing and enriching this information with a proprietary methodology, this entity is able to identify anomalous behaviours in the traffic and discern with very high accuracy between benign and malign behaviours related to security issues. The malign behaviours detected and the data related to them are forwarded to the Anomaly Classifier entity.
- 4) Anomaly Classifier: Once a malign anomalous behaviour is detected by the Alarm Correction entity, the Anomaly Classifier entity is in charge of identifying the type of the anomaly. To that end, the Anomaly Classifier entity uses the measurements and statistics as input for a machine learning technique that is able to classify the anomaly between different volumetric attacks (e.g., DDoS) and zero-day attacks. All the resulting information is then provided to the Remediation module, that based on the characteristics and the anomaly type is able to better provide accurate counter-measurements to mitigate the security issues.

3.4.2.3 The procedure from data acquisition to anomaly detection

The process of anomaly detection is divided in two different phases. In phase one, information is collected from the monitoring vNSFs. This information is stored in the Distributed File System subcomponent for further analysis. In the case of real-time analytics, the information is loaded into cache instead of stored in hard disks, but the workflow is identical. As already explained, the Data Transformation subcomponent can either be distributed, where data is transformed to a generic format (e.g. CSV) before being

sent to the central engine or centralised, so specific formats (e.g. PCAP files) are sent to the central engine (Figure 3.11).

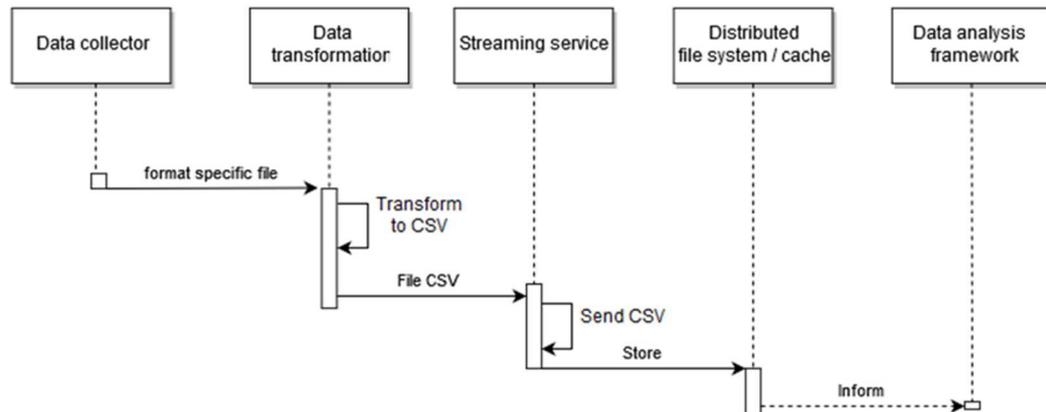


Figure 3.11 Flow diagram of the data acquisition phase

Once the data has been loaded into the distributed file system (either in batch or real time) the next phase (Figure 3.12) implies that the data analysis framework will use the machine learning algorithms to detect anomalies and inform the remediation and recommendation subcomponent. At its turn, this subcomponent will use this information to provide recommendations to the user in the form of specific network services with the configurations and the locations to be deployed.

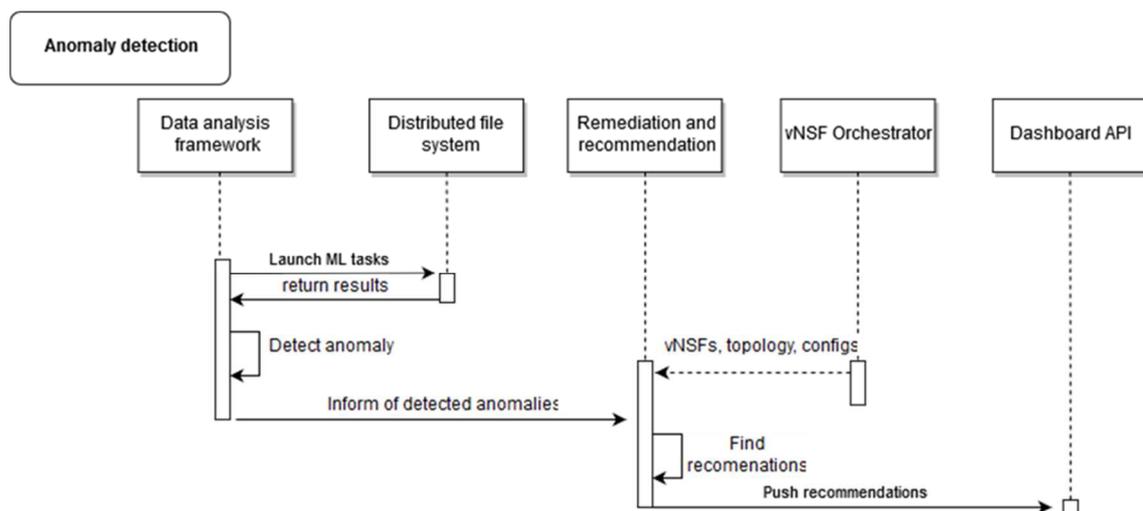


Figure 3.12: Flow diagram of the anomaly detection phase.

3.4.3 The Remediation Phase

The DARE includes two subcomponents on top of the data analytics framework which are in charge of defining the remediation and recommendations actions to be presented to the final user via the Dashboard. A Remediation action consists in a cybersecurity topology responsible for addressing a specific network security threat. These actions will ultimately be translated into a set of vNSFs with proper configuration and deployment location, therefore allowing its instantiation in the secure network environment. Configurations are specified as a set of high-level, technology-independent policies with a uniform description regardless of the targeted vNSF type or implementation.

The recommendation and remediation subcomponent is aware of the current state of the network infrastructure in order to optimise the security impact of the vNSFs of the different Network Services. This awareness is built upon information regarding running instances for both vNSFs and NSs retrieved from the Orchestrator per tenant. A cybersecurity topology will be generated by a detected threat, which is converted into a high-level abstraction of a remediation recipe. However, the actual remediation is not to be performed directly in this subcomponent, which is not be in charge of directly modifying the status of the infrastructure, but to be proposed to the SHIELD operator via the Security dashboard (using the Dashboard API), that is in charge of accepting or declining it. If accepted, the remediation action is applied in the network infrastructure of the tenant through the Orchestrator, which also forwards the request for the translation of policies to low-level configurations, carried out within each vNSF.

Although not of high relevance to the scope of this thesis, a description of the remediation component was considered useful to further illustrate the last functionality of typical anomaly detection systems; security response. The high-level block diagram representing the architecture of the recommendation and remediation engine and its interaction with other components is depicted in the following figure (Figure 3.13) and its internal subcomponents are described as follows:

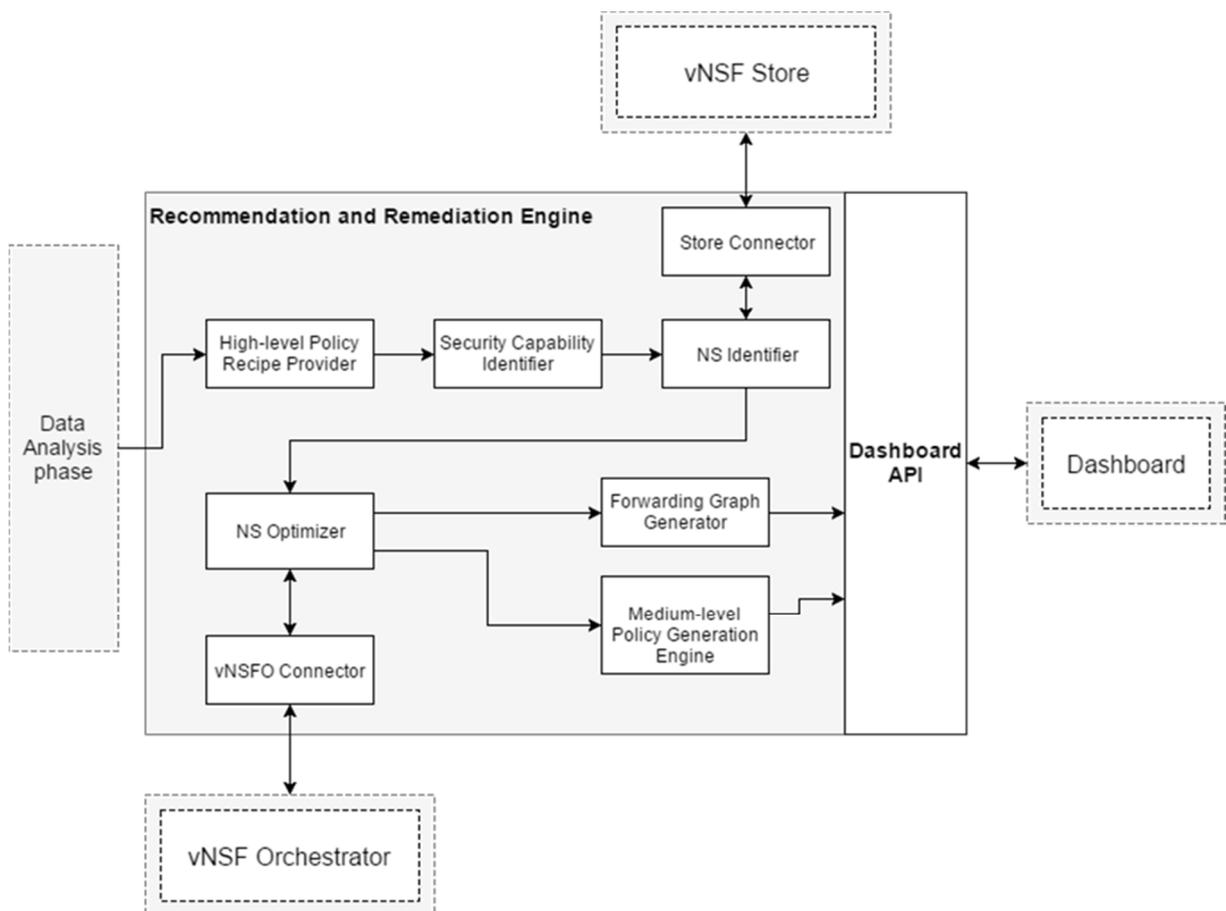


Figure 3.13: The recommendation and remediation subcomponents

- 1) High-level Policy Recipe Provider: This module receives an event from the data analytics framework of the DARE, containing alerts and contextual information related to an occurring threat. Starting from it, this module will define a “recipe”, consisting of a set of security requirements specified in a high-level policy abstraction targeting the mitigation of the detected threat. The

recipes will be stored in an internal repository, in order to allow their update/addition/removal by a security analyst.

- 2) **Security Capability Identifier:** Using the information created by the previous module, this module will be responsible for mapping each remediation high-level recipe to a set of security capabilities. A capability is defined as a basic feature that can be configured to enforce a security action (e.g. address translation, authentication, data protection, authorisation, routing, resource protection, resource analysis).
- 3) **NS Identifier:** This module is in charge of selecting the sets of vNSFs (the NSs) that match the required security capabilities. To do so, it requires the knowledge of the security capabilities offered by each of the NS in the NS Catalogue, hence it directly interacts with the vNSF Store.
- 4) **Store Connector:** This module is in charge of connecting the subcomponent with the vNSF Store, by consuming its API in order to retrieve the information about the available NSs in the catalogue.
- 5) **NS Optimiser:** This module will be in charge of selecting the best NS that matches the required capabilities, according to an optimisation criterion. In addition, this module will be able of verifying if a NS has been already deployed for the tenant, in order to optimise the deployment of the remediation by providing only the updated configuration.
- 6) **vNSFO Connector:** This module is in charge of connecting the subcomponent with the vNSF Orchestrator, by consuming its API in order to retrieve information about the already deployed NSs for the tenant.
- 7) **Forwarding Graph Generator:** This module is in charge of describing the list of vNSFs into a topological arrangement that would allow the vNSF Orchestrator to deploy them into the network infrastructure.
- 8) **Medium-level Policy Generation Engine:** This module generates the medium level policy abstraction starting from the Forwarding Graph and the list of capabilities identified to address the security threat. Each vNSF capability is associated to a policy, expressed in an application-independent syntax.

Finally, the Dashboard is in charge of enabling users and third party applications to use the engine's internal features. Dashboard is therefore the entry point of the solution, seamlessly encapsulating the access and use of all its information and features in this

component. Being the only point of access for the users, it facilitates the integration and builds a more secure application, since the access control is more robust and protected. Besides integrating with all the other components, Dashboard is also responsible for the implementation of a set of support features. It provides user and tenant management features, billing and monetisation capabilities as well as a remediation subcomponent responsible for persisting and dispatching (upon validation by authorised users) all DARE's remediation suggestions.

In this chapter, the architectural aspects of a modern cybersecurity framework were introduced. The profile of a typical cybersecurity engine based on bibliography was presented and then was corresponded with a real state-of-the-art framework that utilises a number of components to perform security analytics. Many of the involved components play an auxiliary (but necessary) role in the anomaly detection procedure, thus they were presented, but will not be analysed any further. The next chapter will focus only on the implementation, configuration and utilisation of the most relevant module for this thesis, the Cognitive Data Analysis module along with its required data ingestion and transformation mechanism.

Chapter 4

Implementation of the cybersecurity analytics engine

4.1 Specifications and used technologies

In the previous chapter the overall design and architecture of the Data Analytics and Remediation Engine (DARE) was presented. This chapter will be devoted to the specification and implementation details of the engine, especially regarding its cognitive Data Analytics module, which comprises the core of the platform and the main scope of this thesis. It was also deemed necessary at this point to provide relevant information for data ingestion as well, since the data acquisition and preparation process is tightly bound to the analytics process. In general, only the implementation of the platform's components that are relevant to the anomaly detection procedure will be presented here (namely data ingestion and cognitive analytics), since the presentation of the rest would be considered irrelevant and would introduce unnecessary complexity to the reader.

Apache Spot² is used as the main technology for the Data Acquisition and the Cognitive Data Analytics modules. Spot is an in-development (at the time of writing this thesis), open-source platform for network telemetry and anomaly detection that provides tools to accelerate the ability to expose suspicious connections and previously unseen attacks using flow and packet analysis technologies. It also features a built-in ingestion subcomponent that is responsible for handling and transferring the raw network data into the data analytics engine. Spot can be considered a superset of state-of-the-art frameworks, as it features a

²Apache Spot page: <http://spot.incubator.apache.org> Accessed July 2017

combination of distributed computing, data lake management and machine learning frameworks along with auxiliary services to ingest, analyze and present the detected anomalies in network traffic. It is built over very mature technologies, more specifically:

- Cloudera CDH³ for data ingestion and storage (which uses Hadoop HDFS and Apache Hive). CDH is an Open Source platform distribution that helps to perform end-to-end Big Data workflows.
- Apache Spark⁴ for machine learning and streaming. Apache Spark is a fast and general engine for large-scale data processing.
- ReactJS⁵ and Flux⁶ for the web components. ReactJS is a Javascript framework for building user interfaces. Flux is the application architecture that Facebook uses for building client-side web applications.
- IPython⁷ for the Spot virtualization server. IPython is a command shell for interactive computing in multiple programming languages.
- GraphQL⁸ for query data from HDFS Parquet files. GraphQL is a query language for custom APIs, and a server-side runtime for executing queries by using a user-defined type system.
- Apache Hadoop⁹ for distributed file system. Hadoop allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- Apache Hive¹⁰ for data storage. Hive facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.

³ Cloudera Distribution for Hadoop page: <https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html> Accessed July 2017

⁴ Apache Spark page: <https://spark.apache.org/> Accessed July 2017

⁵ React JS page: <https://facebook.github.io/react/> Accessed July 2017

⁶ Flux page: <https://facebook.github.io/flux/> Accessed July 2017

⁷ IPython page: <https://ipython.org/> Accessed July 2017

⁸ GraphQL page: <http://graphql.org/learn/> Accessed July 2017

⁹ Apache Hadoop page: <https://hadoop.apache.org/> Accessed July 2017

¹⁰ Apache Hive page: <https://hive.apache.org/> Accessed July 2017

The installation and utilization steps of Apache Spot for anomaly detection purposes will be thoroughly presented in the following subsections. In Figure 4.1, all related subcomponents are shown together with the technology that was used to develop them. It is worth to mention that Spot is not “yet another IDPS” but a platform that offers to the possibility to develop one’s own machine learning processes. This approach encourages the focusing of the effort on the innovative algorithms that will can used for threat detection and mitigation in future work.

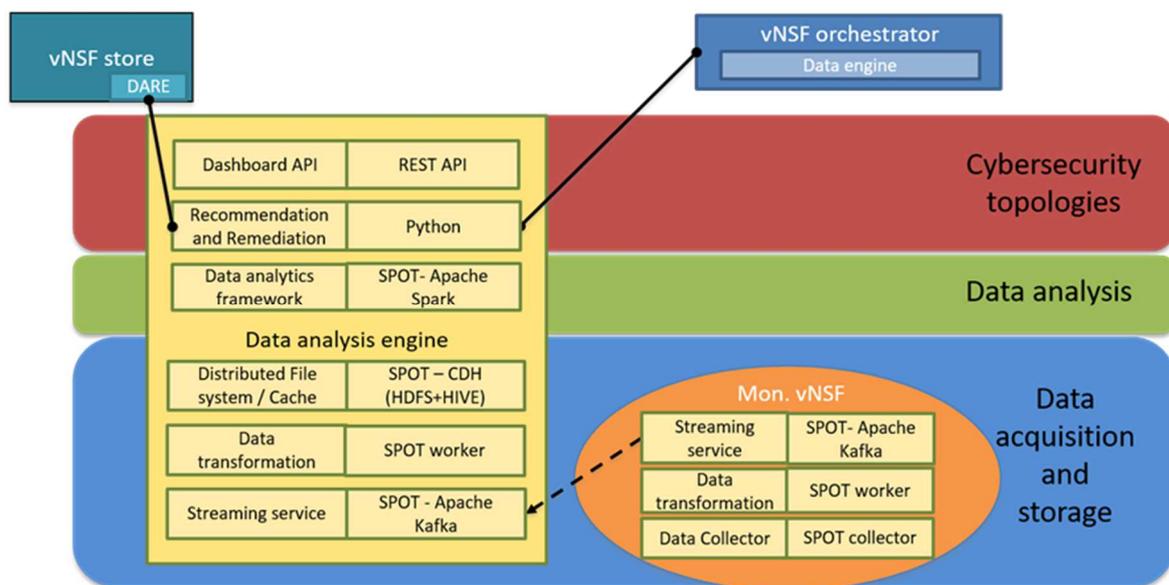


Figure 4.1: List of subcomponents and used technologies

4.2 Implementation of the data ingestion process

As mentioned in Chapter 3, DARE encompasses the option of both a centralised and a distributed data ingestion architecture. The centralised architecture of the Data Acquisition and Preparation phase, as depicted in Figure 3.5 of the previous chapter, features a built-in ingestion framework of the Apache Spot platform, which is responsible for handling the raw network data that are transferred directly to a specific path of the data analytics engine. Ingestion framework consists of a number of edge nodes, running on

Linux OS and handling the incoming network traffic. It also supports Apache Kafka¹¹ as a streaming platform, for handling all the real-time network data feeds.

Inside the ingestion framework, daemons called data collectors are operating, which monitor the path for new files with network data generated by monitoring vNSFs. In the case of centralised architecture, once new data files are detected, collectors capture them from local file system and publish them to Apache Kafka. Apache Kafka splits the raw network data into specific topics and smaller partitions, while creating a data pipeline for each type of data. Each pipeline sends the network data, stored by data collectors, to specific daemons, called workers. Workers are running in the background inside the data analytics engine, as part of the data transformation subcomponent. Each worker is subscribed to a specific topic and partition of Apache Kafka. It reads raw network data from the partition, decodes and translates it into comma-separated files (CSV), by using dissection tools. Once the data has been transformed, worker stores the input in HDFS with both the original and human-readable format, making it available to Hive tables, so it can be accessible by SQL queries. This procedure is shown in Figure 4.2:

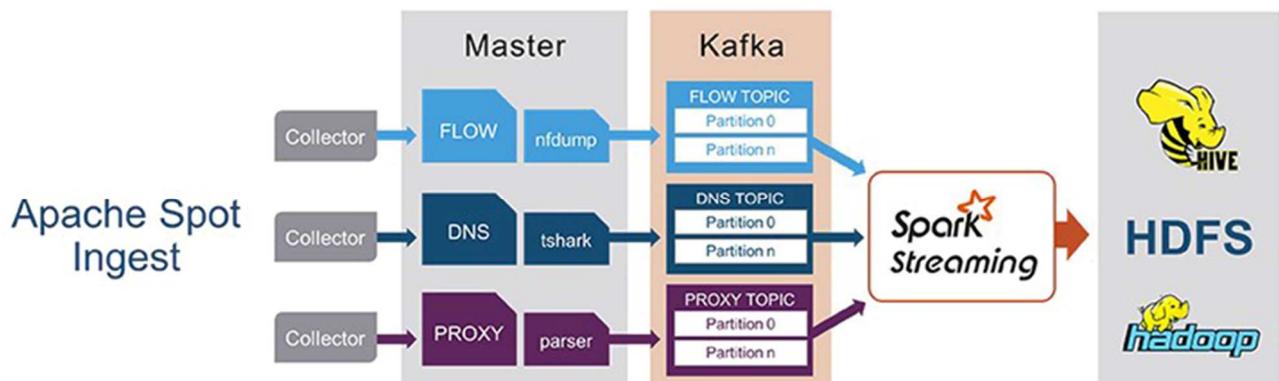


Figure 4.2: Dataflow of the ingestion framework

¹¹ Apache Kafka page: <https://kafka.apache.org/> Accessed July 2017

This phase also supports a distributed architecture that was shown in Figure 3.6. The main difference between the centralised and the distributed architecture is that in the centralised one, workers are running inside the data analytics engine, while in the distributed one, there is one worker per vNSF. Workers will decode and translate the raw network file into a comma-separated file with specific structure, and will send it to the Apache Kafka streaming platform. Kafka will transfer the text file to the data analytics engine, where the file will be imported directly into Hive tables. Unlike the centralised architecture, which stores both the original and the human-readable formats inside the data analytics engine, only the transformed version of the files is stored in the distributed architecture. Since both centralised and distributed ingestion architectures are identical in terms of output, meaning that they provide the same input to the data analytics component it was considered preferable for this thesis to focus on one of them, more specifically on the built-in centralized architecture.

As the data ingestion is based on the Apache Spot platform, the heterogeneous network information that are captured via monitoring vNSFs are compatible with the following structure and format:

- **Netflow:** .nfcapd data files which contain network traffic as it enters or exits from a monitoring interface. By analysing netflow data, a network administrator is able to determine things such as the source and destination of traffic, class of service and the causes of congestion.
- **DNS:** .pcap files (packet capture) out of DNS servers. These files contain network packet data, created during a live network capture. By analysing .pcap files, vital information can be retrieved regarding the monitored network and its characteristics.
- **Proxy:** bluecoat proxy format logs are supported.

Samples of the aforementioned data types are included in the Appendix.

By default, Apache Spot uses specific open-source decoders in order to transform raw network data into comma-separated files (.csv). According to the data type, a different dissection tool is being used. For flow traffic, a modified version of nfdump is used to dissect the flow packets into comma-separated files. For PCAP files (packet captures) the TShark tool is used with a combination of options so as to generate the comma-separated files. The proxy log files are parsed using Apache Streaming before being inserted into Hive tables. The comma-separated files are used as input to the data analytics process that will be shown subsequently.

4.3 Implementation of the data analytics process

The data analytics process could be vaguely described as the processing of data that were acquired via ingestion using machine learning techniques for the extraction of useful information. As illustrated in Figure 3.7, the platform leverages two different data analytics modules that export their findings to a shared mitigation engine, in order to produce optimal results. The Cognitive Data Analytics module that will be based on the Apache Spot platform will be described in this subsection, while the Security Data Analysis module which was introduced in the previous chapter implements a version of proprietary network visibility software that will be considered a “black box” and will not be further disclosed.

The Cognitive Data Analytics module infrastructure consists of a cluster of nodes running on open-source OS and virtual machines. Each node performs a number of operations and is orchestrated by cluster and resource management technologies, specifically designed for Big Data applications. The specifications of the Cognitive Data Analytics module include:

- A cache system for real time analytics.
- Unified services for orchestration, operations and resource management.

- Machine Learning modules that provide scalable ML algorithms for network traffic filtering, as part of a cluster-computing framework.
- An Operational Analytics module that allows for the implementation of whitelisting and filtering techniques that will help reduce false positives and offer remediation recommendations to the users.

The module consists of a number of nodes (physical or virtual machines), each of them performing a designated task. The worker nodes receive the ingested network traffic from the Data Acquisition Phase and are responsible for the operation of the machine learning entity. The Operational Analytics nodes execute the filtering and whitelisting functionalities and are intended to operate as the final editing step, before pushing the detection results to the Remediation Engine.

The cognitive Data Analytics module performs anomaly detection analytics, following the Spot architecture, by deploying a collection of state-of-the-art technologies (Hadoop, Spark, Kafka etc.) in the form of an integrated ecosystem, the Cloudera Distribution for Hadoop. CDH is an Apache-licensed open-source framework that delivers the core elements of scalable storage and distributed computing, along with a Web-based UI and enterprise capabilities and is considered the most popular distribution for Apache Hadoop and related projects. Spot utilises CDH as a general dependency package for the development of its three main parts: ingestion, machine-learning and operational analytics.

The machine-learning performed by Spot is responsible for the detection of anomalies in network traffic and the prevention or mitigation of potential threats. The engine contains routines for performing anomaly detection on netflow, DNS and proxy logs. These routines consume a collection of network events and produce a list of the events that are considered to be the least probable, these being considered the most suspicious. The statistical model that is currently used for discovering abstract topics of these events and ultimately discovering normal and abnormal behaviour is a topic modelling algorithm called Latent Dirichlet Allocation which was thoroughly presented in §2.3.2.4. LDA is used here

as a generative probabilistic model used for discrete data that is applied to network traffic by converting network log entries into words through aggregation and discretisation, in order to discover hidden semantic structures. The next subsection §4.3.1 is dedicated to Spot's LDA implementation for analysing suspicious connections. Spot executes LDA routines using a Scala Spark implementation from MLlib, Apache Spark's scalable machine learning library. It should be noted that Spot's current capabilities do not include any anomaly classification algorithms that would interpret the detected outliers as specific threats/attacks, thus the user can only detect malicious behaviour without identifying it as a particular type of attack.

Spot also includes an operational analytics entity that is used to exploit the results of the machine-learning process. It uses the Jupyter/iPython notebook, a server-client application that allows editing and running notebook documents via a web browser, in order to apply filtering and whitelisting services thus providing a more accurate view of the overall anomaly detection procedure, by reducing false-positives.

4.3.1 Latent Dirichlet Allocation implementation for network anomaly detection

This subsection is dedicated to the presentation of the machine learning algorithm that constitutes the basis of the anomaly detection process. It is used to support a family of suspicious connections analyses that identify the most suspicious or unlikely network events in the observed network and report these to the user for further investigation to determine if they are indicative of maliciousness or malfunction. If the user determines that the connection is acceptable behaviour and has been wrongly classified, the operational analytics entity of the Spot engine allows the flagging of the event as acceptable so that a new model can be generated that will not flag similar events. The suspicious connects

analysis is a form of unsupervised anomaly detection that uses topic modelling to infer common network behaviours and build a model of behaviour for each IP address, using a Spark-MLlib implementation of Latent Dirichlet Allocation (LDA). Here, the mathematical principles behind the suspicious connections analysis are briefly analysed. For more information on LDA in general, see §2.3.2.4.

As mentioned before, Spot supports analyses on the following data sources; log entries that occur from which will be referred to as “network events” from now on:

- Netflow logs
- DNS logs
- HTTP Proxy logs

The suspicious connects analysis infers a probabilistic model for the network behaviours of each IP address. Each network event is assigned an estimated probability (henceforth, the event’s “score”). Those events with the lower scores are flagged as “suspicious” for further analysis. The probabilistic model is generated via the Latent Dirichlet Allocation topic modelling algorithm, a technique from natural language processing that analyses a collection of natural language documents, and infers the topics discussed by the documents. Firstly, the probability distributions that arise from an LDA model are described along with how anomaly scores can be assigned to words of a document. Subsequently, the formation of ‘words’ and ‘documents’ from network logs is presented so that a network log entry is provided an anomaly score given by the score of the word to which it is associated.

The input and output of the LDA implementation are:

- **Input:** A collection of documents, each viewed as a multiset of words (bag of words). An integer k which is the number of latent topics for the model to learn.

- **Output:** Two families of distributions. For each document, a “document’s topic mix” which gives the probability that a word selected at random from the document belongs to any given topic (that is, the fraction of that document dedicated to any given topic). For each topic, a “topic’s word mix” which gives the probability of any given word conditioned on the topic (that is, the fraction of that topic dedicated to each word).

and the probability distributions in mathematical notation:

$p(z \theta_d)$	Probability of topic z given document d
$p(w z)$	Probability of word w given topic z

Table 4.1: Probability distributions used by LDA

An assumption is made that a topic’s word mix is independent of the document in question, so that it’s possible to perform a model-estimate of the probability of a word, w , appearing in the document, d , as follows:

$$p(w|\theta_d) = \sum_z p(w|z)p(z|\theta_d) \tag{12}$$

By viewing the logged behaviour of an IP address as a document (eg. all DNS queries of a particular client IP) and the constituent log entries as “words” it is straightforward to apply topic modelling to analyse network traffic using the following transformation:

Text Corpora	Network Logs
document	log records of a particular IP address
word	(simplified) log entry
topic	profile of common network behaviour

Table 4.2: Correspondence between text corpora as seen on LDA mathematical principles and network logs used by Spot’s LDA implementation

However, LDA (and topic modelling in general) comes with an interesting peculiarity that needs to be taken into consideration: for topic modelling to provide interesting results,

there should be significant overlap in the words used by different documents, whereas network log entries contain nearly unique identifiers such as network addresses and timestamps. For this reason, in order to perform topic modelling on network events, the log entries must be simplified into words.

The conversion of network events into words is the novelty of Spot's anomaly detection procedure. The process must preserve enough information to turn up interesting anomalies during malicious behaviour or malfunction, it must create words with enough overlap between documents (IP addresses) so that the topic modelling step produces meaningful results and it should distil information that is particular to the "type" of traffic rather than a specific machine (to justify the simplifying assumption made to estimate word probabilities). Next follows the word creation procedure for the 3 network data sources types:

Netflow logs

A netflow record is simplified into two separate words, one to be inserted in the document associated to the source IP and another (possibly different word) inserted into the document associated to the destination IP. The words are created as follows:

Feature (string 'letter in the word')	Type
<p>1. Flow Direction:</p> <ul style="list-style-type: none"> If both ports (between source and destination) are 0, then this feature is missing from the words that go into both the source and destination IP documents If exactly one port is 0, this feature is missing for the IP document associated to the 0 port, and this feature is given as "-1" for the IP document associated to the non-zero port 	Heuristic

<ul style="list-style-type: none"> • If neither port is zero, and either both or neither of the listed ports are strictly less than 1025, this feature is missing for both source IP and destination IP words. • If neither port is zero and only one of the ports is strictly less than 1025, this feature is given as “-1” for the IP document associated with port that is less than 1025 and is missing for the IP document associated to the other (high) port. 	
<p>2. Key Port:</p> <ul style="list-style-type: none"> • If both source and destination ports are 0 this feature is given as “0” for both source and destination IP documents. • If exactly one of the ports is non-zero this feature is given as the non-zero port number for both source and destination IP documents. • If exactly one port is less than 1025 and this port is not zero, this feature is given as this port number for both the source and destination IP documents. • If both ports are non-zero and strictly less than 1025 this feature is given as “111111” for both the source and destination IP documents. • If both ports are greater than or equal to 1025 this feature is given as “333333” for both the source and destination IP documents. 	<p>Heuristic</p>
<p>3. Total Bytes</p>	<p>Binned into Deciles</p>
<p>4. Time of day</p>	<p>Binned into Deciles</p>
<p>5. Number of Packets</p>	<p>Binned into Quintiles</p>

Table 4.3: Word creation for netflow logs

In essence, the topic modelling algorithm for netflow logs (which is similar for the other two types of network data as well) considers a single netflow record as a word, a collection of records regarding the same IP as a document and the probability distribution of records (words) as a topic that represents a network profile. The procedure is briefly defined by the following steps:

1. Convert netflow records into words (one for each IP)
2. Treat netflow records about each IP as documents of these words.
3. Infer a collection of "topics" that represent common profiles of network traffic. These "topics" are probability distributions on words.
4. Each IP has a mix of topics corresponding to its behaviour.
5. Estimate the probability of a new word by converting its netflow record into a word, then combining the word probabilities per topic using the topic mix of the particular IP.

To better understand the word creation procedure, some examples are provided:

1. A record with source port 1066, destination port 301, bytes transferred falling within the lowest 10% of all observed values for the given batch of data, time of day falling within the top 10% of all observed values, with number of packets falling within the lowest 25% of all observed values.
 - The word "301_0_9_0" is created for the source IP document.
 - The word: "-1_301_0_9_0" is created for the destination IP document.
2. A record with source port 1194, destination port 1109, with bytes transferred falling between the 10th and 20th percentiles of all observed values for the given batch of data, time of day falling within the top 10% of all observed values, and packet count within the top 20% of all observed values.
 - The word: "333333_1_9_4" is created for both the source and destination IP documents.

DNS logs

A DNS log entry is simplified into a word and inserted into the document associated to the client IP making the DNS query. The word is created as follows:

Feature(string 'letter in the word')	Type
1. Analyze DNS query name: <ul style="list-style-type: none"> • If belongs to Alexa top 1 million list, use “1” • If belongs to user domain, use “2” • Otherwise, use “0” 	Heuristic
2. Frame length	Binned into Deciles
3. Time of day	Binned into Deciles
4. Subdomain Length	Binned into Quintiles
5. String Entropy of Subdomain	Binned into Quintiles
6. Number of periods in Subdomain	Binned into Quintiles
7. DNS query type	As given in netflow record
8. DNS query response code	As given in netflow record

Table 4.4: Word creation for DNS logs

Proxy logs

A proxy log entry is simplified into a word and inserted into the document associated to the client IP making the proxy request. The word is created as follows:

Feature(string 'letter in the word')	Type
1. Analyze DNS query name: <ul style="list-style-type: none"> • If belongs to Alexa top 1 million list, use “1” • If belongs to user domain, use “2” • Otherwise, use “0” 	Heuristic
2. Time of day	The hour part of the timestamp

3. Request Method	As given (eg. “GET”, “POST” etc.)
4. String Entropy of URI <ul style="list-style-type: none"> Use the string for the bin number (0-18) into which the entropy value falls, using bins defined by the following cut-off values: (0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3.0, 3.3, 3.6, 3.9, 4.2, 4.5, 4.8, 5.1, 5.4, 20) 	Binned
5. Top level content type	String (eg. “image”, “binary”)
6. Frequency of user agent type in training data <ul style="list-style-type: none"> Use the string for the bin number (0-∞) into which the entropy value falls, using bins defined by the following cut-off values: (0, 1, 2, 4, 8, 16, 32, ...) 	Binned
7. Response code	As given

Table 4.5: Word creation for proxy logs

Using the above LDA implementation, the Cognitive Data Analytics module is capable of identifying a wide range of malicious network behaviour such as recon attacks (advanced scanning), probing on services (e.g. SSH logon attempts), successful connections to these services (e.g. rare SSH connections from outside), botnet/beaconing, denial of service attacks, data exfiltration (a large amount of traffic going somewhere unfamiliar) etc. These are depicted as network events with a low probability score compared to normal network activity, meaning that they are unlikely to occur and -therefore- are suspicious.

4.3.2 Configuration of the anomaly detection framework

In this subsection, the guidelines for the installation and configuration of the Spot engine are presented. Although the procedure regarding the installation and configuration of the testbed and the engine's prerequisites is described here, the comprehensive set of instructions that includes all the necessary installation steps can be found in the Appendix.

Apache Spot is not a standalone application as it requires prerequisite software properly installed, configured and distributed to a number of nodes. The engine itself is also decomposed into a number of components, each of them featuring different functionalities that must be placed into separate nodes. It must therefore be installed on a new or existing Hadoop cluster, with its required components viewed as services and distributed according to common roles in the cluster. The recommended approach is to follow the Apache Community validated deployment of Hadoop, the Cloudera Distribution for Hadoop (CDH) as shown in Figure 4.3. Each rectangle of the figure signifies a cluster node, which can be either a physical machine or a virtual one (e.g. VM, LXC, Docker etc.). Although the installation documentation suggests the utilisation of five nodes, a minimum of three nodes is actually required in order to setup the necessary components, namely Ingestion (IN), Machine Learning (ML) and Operational Analytics (OA). More specifically, one node (Edge) can be created for Ingestion, one node (Worker) for Machine Learning and one (Master) for Operational Analytics that includes all services of Master 1, Master 2 and Cloudera Manager nodes of Figure 4.3.

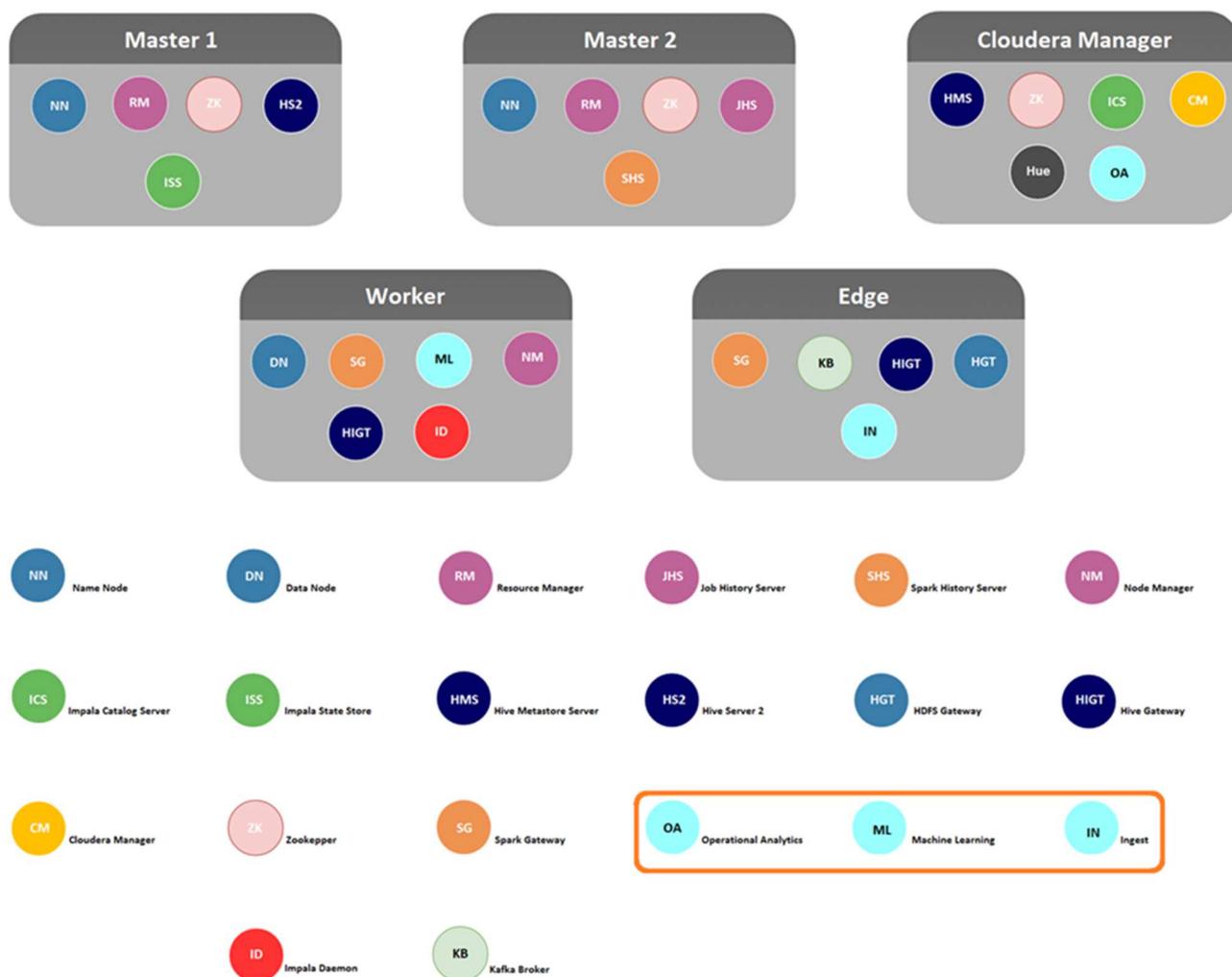


Figure 4.3: Recommended node configuration of the CDH cluster

The installation of CDH as a prerequisite for the installation of Spot offers the core elements of Hadoop, scalable storage and distributed computing, along with a Web-based user monitoring interface the Cloudera Manager. It facilitates the installation of all the necessary technologies by using parcels, a binary distribution format containing the program files, along with additional metadata used by Cloudera Manager. Parcels are self-contained and installed in a versioned directory, meaning that they can be automatically downloaded, distributed, and activated for each node of the cluster upon request.

In our installation, we configured three virtual nodes on a single physical machine by using Linux Containers (LXCs), an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel. The specifications of the physical machine include a 4-core Intel Xeon E3-1225 processor clocked at 3.20GHz, 32GB of DDR3 RAM, and a 3TB hard disk drive. The three hosts are running Ubuntu Trusty 14.04. After the creation of the LXCs, the Cloudera Quickstart VM Cluster version 5.11.1y was used to install the prerequisite components of Apache Spot as seen in Figure 4.3, namely: HDFS, Hive, Impala, Kafka, Spark, Yarn and Zookeeper using the documentation found at Cloudera's webpage¹².

The entirety of the Apache Spot installation files is divided into the following four components that must be placed inside the three nodes:

- **spot-setup** — scripts that create the required HDFS paths, hive tables and configuration for apache spot (incubating).
- **spot-ingest** — binary and log files are captured or transferred into the Hadoop cluster, where they are transformed and loaded into solution data stores.
- **spot-ml** — machine learning algorithms are used to add additional learning information to the ingest data, which is used to filter and sort raw data.
- **spot-oa** — data output from the machine learning component is augmented with context and heuristics, then is available to the user for interacting with it.

Our setup with the 3 containers (LXCs) was the following:

- ❖ **Cloudera-host-1** node was configured as the Machine Learning node, including **spot-ml**.

¹² CDH 5.11.y documentation: <https://www.cloudera.com/documentation/enterprise/5-11-x.html>

- ❖ **Cloudera-host-2** node was configured as the Setup and Edge Node, including **spot-setup** and **spot-ingest**.
- ❖ **Cloudera-manager** node was configured as the Operational Analytics node, including **spot-oa**.

In Figures 4.4 and 4.5 the Cloudera Manager interface is shown, depicting the three nodes after the installation of the prerequisites. The step-by-step installation of the spot-setup, spot-ingest, spot-ml and spot-oa components is a tedious procedure that is described in the Appendix.

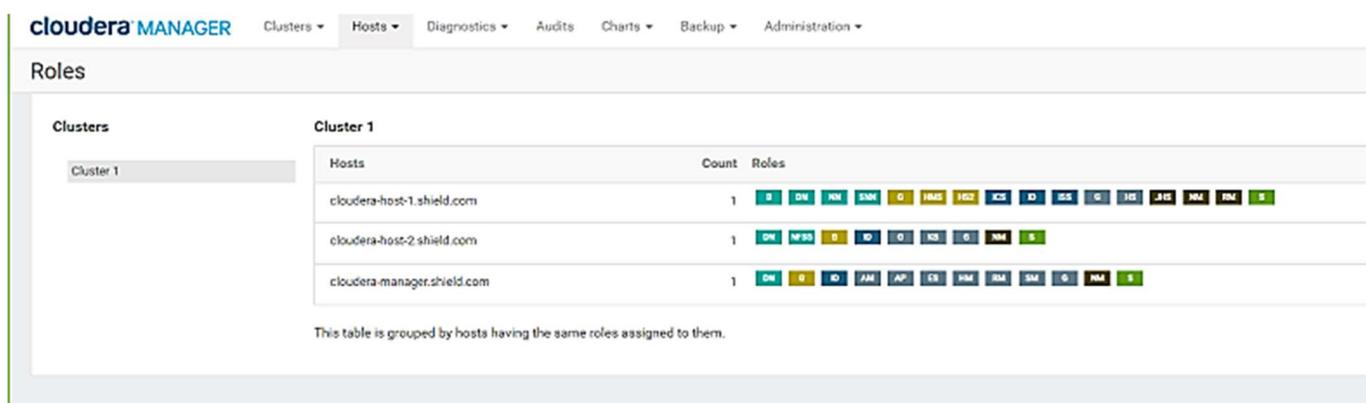


Figure 4.4: Roles of the cluster nodes after the CDH configuration

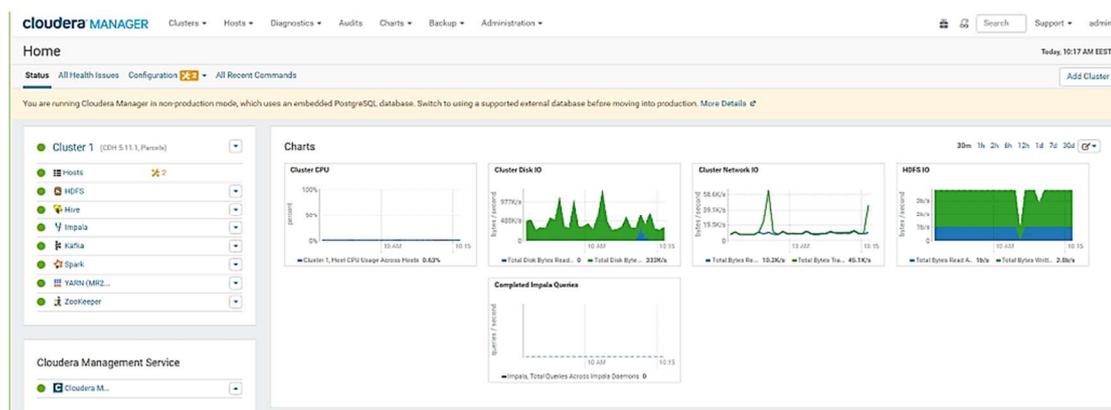


Figure 4.5: The Cloudera Manager Interface, showing the cluster health and usage metrics

In the above deployment, the Ingest component runs on an edge node, which is the expected use of this role. The Operational Analytics component runs on a node intended for browser-based management and user applications, so that all user interfaces are located on a node with the same role. The Machine Learning component is installed on one worker node, but can be easily expanded to more nodes as the resource management for an ML pipeline is similar for functions inside and outside Hadoop.

4.4 Example use of the cybersecurity engine

After the installation of the required components and prerequisites, the user is able to feed the ingestion framework with network traffic logs (netflow, DNS, proxy), run machine-learning scripts on the ingested data and monitor the results of the algorithms via a browser-based GUI. It should be noted that by the time of writing this thesis, the engine did not come with a graphical UI (apart from the monitoring browser-based GUI), so any actions regarding its utilisation from the launching of the collectors and workers to the running of anomaly detection scripts has to be performed via terminal. In this section, we will demonstrate the full action cycle from the acquisition of a netflow dataset to the execution of the machine-learning script (LDA implementation) and the extraction of the analysis results. The procedure is almost identical for DNS and proxy datasets too, besides from a few different command parameters that are thoroughly explained in Spot's documentation webpage.

4.4.1 Acquisition and ingestion of netflow data

In order to initiate the ingestion procedure, a suitable network traffic log must exist at Apache Spot's predefined ingestion path of the edge node for this specific data type. In the case of netflow data, we need .nfcapd files that were either captured in the past (e.g. by a firewall) or are being concurrently captured by the edge node and put to the netflow

ingestion path. For the latter, the command to initiate the capture of netflow data to a specified filepath is:

```
$ nfcapd -w -T all -l /home/ubuntu/incubator-spot/spot-ingest/pipelines/flow
```

The above command will start capturing .nfcapd files and send them to the filepath that handles netflow logs exclusively (other filepaths are created for other file types). Next, the ingest master collector that will create a specific Kafka topic must be launched, using the following command at the `~/.../spot-ingest` level:

```
$ nohup python master_collector.py -t flow -w 1
```

The above command will start the master collector for netflow data and will create a topic with a specific name (e.g. SPOT-INGEST-flow_09_53_41_322692). In case of DNS or proxy datasets, the `flow` parameter must be switched to `dns` or `proxy` respectively. We can then start a netflow worker and have it listen to that topic created by the master:

```
$ nohup python worker.py -t flow -i 0 --topic SPOT-INGEST-flow_09_53_41_322692
```

Again, the `flow` parameter can be changed for other data types. We can always monitor that the ingestion procedure is active by reading the `nohup.out` file that has been created at the same level by typing:

```
$ tail -f nohup.out
```

The `nohup.out` log informs the user of the detection of a new file, its processing and transfer from the local filesystem to the HDFS and Hive, as shown in Figure 4.6:

```

2017-08-24 13:55:01.907 - SPOT_INGEST.WRK.DNS - INFO - ----- New File received -----
2017-08-24 13:55:01.908 - SPOT_INGEST.WRK.DNS - INFO - File: /user/spotuser/dns/binary/20170824/13/temp_00004_20170824134816_spot_00000_20170824134815.pcap
2017-08-24 13:55:01.909 - SPOT_INGEST.WRK.DNS - INFO - Getting file from hdfs: hadoop fs -get /user/spotuser/dns/binary/20170824/13/temp_00004_2017082413481
6_spot_00000_20170824134815.pcap /home/spotuser/traffic/dns/local_staging/
2017-08-24 13:55:01.910 - SPOT_INGEST.WRK.DNS - INFO - SPOT.Utils: Executing: hadoop fs -get /user/spotuser/dns/binary/20170824/13/temp_00004_20170824134816
_spot_00000_20170824134815.pcap /home/spotuser/traffic/dns/local_staging/
2017-08-24 13:55:06.082 - SPOT_INGEST.WRK.DNS - INFO - Processing file: tshark -r /home/spotuser/traffic/dns/local_staging//temp_00004_20170824134816_spot_0
0000_20170824134815.pcap -E separator=, -E header=y -E occurrence=f -T fields -e frame.time -e frame.time_epoch -e frame.len -e ip.src -e ip.dst -e dns.resp
.name -e dns.resp.type -e dns.resp.class -e dns.flags.rcode -e dns.a 'dns.flags.response == 1' > /home/spotuser/traffic/dns/local_staging//temp_00004_201708
24134816_spot_00000_20170824134815.pcap.csv
2017-08-24 13:55:06.082 - SPOT_INGEST.WRK.DNS - INFO - SPOT.Utils: Executing: tshark -r /home/spotuser/traffic/dns/local_staging//temp_00004_20170824134816
_spot_00000_20170824134815.pcap -E separator=, -E header=y -E occurrence=f -T fields -e frame.time -e frame.time_epoch -e frame.len -e ip.src -e ip.dst -e dn
s.resp.name -e dns.resp.type -e dns.resp.class -e dns.flags.rcode -e dns.a 'dns.flags.response == 1' > /home/spotuser/traffic/dns/local_staging//temp_00004_
20170824134816_spot_00000_20170824134815.pcap.csv
2017-08-24 13:55:06.307 - SPOT_INGEST.WRK.DNS - INFO - Creating staging: hadoop fs -mkdir -p /user/spotuser/dns/stage/550630
2017-08-24 13:55:06.308 - SPOT_INGEST.WRK.DNS - INFO - SPOT.Utils: Executing: hadoop fs -mkdir -p /user/spotuser/dns/stage/550630
2017-08-24 13:55:10.448 - SPOT_INGEST.WRK.DNS - INFO - Moving data to staging: hadoop fs -moveFromLocal /home/spotuser/traffic/dns/local_staging//temp_00004
_20170824134816_spot_00000_20170824134815.pcap.csv /user/spotuser/dns/stage/550630/
2017-08-24 13:55:10.449 - SPOT_INGEST.WRK.DNS - INFO - SPOT.Utils: Executing: hadoop fs -moveFromLocal /home/spotuser/traffic/dns/local_staging//temp_00004_
20170824134816_spot_00000_20170824134815.pcap.csv /user/spotuser/dns/stage/550630/
2017-08-24 13:55:14.799 - SPOT_INGEST.WRK.DNS - INFO - Loading data to hive: hive -hiveconf dbname=spot -hiveconf y=2017 -hiveconf m=08 -hiveconf d=24 -hive
conf h=13 -hiveconf data_location='/user/spotuser/dns/stage/550630' -f pipelines/dns/load_dns_avro_parquet.hql
2017-08-24 13:55:14.799 - SPOT_INGEST.WRK.DNS - INFO - SPOT.Utils: Executing: hive -hiveconf dbname=spot -hiveconf y=2017 -hiveconf m=08 -hiveconf d=24 -hive
conf h=13 -hiveconf data_location='/user/spotuser/dns/stage/550630' -f pipelines/dns/load_dns_avro_parquet.hql

Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/hive-common-1.1.0-cdh5.11.1.jar!/hive-log4j.prope
rties
hiveconf: data_location=/user/spotuser/dns/stage/550630
hiveconf: y=2017
hiveconf: m=08
hiveconf: d=24
hiveconf: h=13
hiveconf: dbname=spot
OK
Time taken: 1.057 seconds
OK
Time taken: 0.178 seconds

```

Figure 4.6: Example of the nohup.out log, providing information during the ingestion process

At this point, files that are captured by the nfcapd daemon are being ingested by the workers, sent to the HDFS as binaries and converted to .csv files in Hive (also sent to the HDFS) that are readable by the machine-learning script.

4.4.2 Machine-learning for netflow data

At this point, the distributed filesystem is populated by .nfcapd binary files as well as their Hive .csv counterpart for a specific date (e.g nfcapd.20170705). To initiate a machine-learning job, user has to go to the `~/.../spot-ml` level of the machine-learning node and execute the following command:

```
$ ./ml_ops.sh 20170705 flow 1e06 10000
```

The above will start the anomaly detection LDA implementation of the engine for netflow files of that specific data, outputting scores that have a suspicious threshold of $1e06$ or lower and will display only the top 10000 results. All parameters are customisable for different data types, thresholds (setting the cut-off value equal to 1 will display all records in a descending suspiciousness order) or number of results (leaving that parameter blank will display all results that meet the threshold). The procedure generates log information throughout its cycle with the following format (Figure 4.7):

```
spotuser@cloudera-host-1:~/spot-ml$ ./ml_ops.sh 20170804 flow 1e06 100
17/08/28 15:48:00 INFO fs.TrashPolicyDefault: Moved: 'hdfs://cloudera-host-1.shield.com:8020/user/spotuser/flow/scored_results/20170804/scores'
to trash at: hdfs://cloudera-host-1.shield.com:8020/user/spotuser/.Trash/Current/user/spotuser/flow/scored_results/20170804/scores
17/08/28 15:48:02 INFO Remoting: Starting remoting
17/08/28 15:48:02 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriverActorSystem@10.101.30.11:34637]
17/08/28 15:48:02 INFO Remoting: Remoting now listens on addresses: [akka.tcp://sparkDriverActorSystem@10.101.30.11:34637]
17/08/28 15:48:11 WARN spark.SparkContext: Dynamic Allocation and num executors both set, thus dynamic allocation disabled.
17/08/28 15:48:16 INFO SuspiciousConnectsAnalysis: Loading data from: /user/spotuser/flow/hive/y=2017/m=08/d=04/
17/08/28 15:48:21 INFO SuspiciousConnectsAnalysis: Starting flow suspicious connects analysis.
17/08/28 15:48:21 INFO SuspiciousConnectsAnalysis: Fitting probabilistic model to data
17/08/28 15:48:21 INFO SuspiciousConnectsAnalysis: Training netflow suspicious connects model from /user/spotuser/flow/hive/y=2017/m=08/d=04/
17/08/28 15:48:37 INFO SuspiciousConnectsAnalysis: 9915.0,19810.0,29003.0,36992.0,44483.0,52287.0,60294.0,68603.0,77015.0,86393.0
17/08/28 15:48:37 INFO SuspiciousConnectsAnalysis: calculating byte cuts ...
17/08/28 15:48:51 INFO SuspiciousConnectsAnalysis: 66.0,72.0,82.0,119.0,131.0,164.0,252.0,313.0,775.0,2.149634827E9
17/08/28 15:48:51 INFO SuspiciousConnectsAnalysis: calculating pkt cuts
17/08/28 15:49:02 INFO SuspiciousConnectsAnalysis: 1.0,1.0,1.0,1.0,1436011.0
17/08/28 15:50:21 INFO SuspiciousConnectsAnalysis: Running Spark LDA with params alpha = 1.02 beta = 1.001 Max iterations = 20 Optimizer = em
[Stage 67:=====] (193 + 2) / 200]17/08/28 15:51:49 WARN netlib.BLAS: Failed to load implementati
on from: com.github.fommil.netlib.NativeSystemBLAS
17/08/28 15:51:49 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
17/08/28 16:06:49 INFO SuspiciousConnectsAnalysis: Identifying outliers
17/08/28 16:06:49 INFO SuspiciousConnectsAnalysis: Netflow suspicious connects analysis completed.
17/08/28 16:06:49 INFO SuspiciousConnectsAnalysis: Saving results to : /user/spotuser/flow/scored_results/20170804/scores
17/08/28 16:09:16 WARN SuspiciousConnectsAnalysis: Saving invalid records to /user/spotuser/flow/scored_results/20170804/scores/invalid_records
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/hive-exec-1.1.0-cdh5.11.1.jar!/shaded/parquet/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/hive-jdbc-1.1.0-cdh5.11.1-standalone.jar!/shaded/parqu
et/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/parquet-format-2.1.0-cdh5.11.1.jar!/shaded/parquet/org
/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/parquet-hadoop-bundle-1.5.0-cdh5.11.1.jar!/shaded/parqu
et/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/parquet-pig-bundle-1.5.0-cdh5.11.1.jar!/shaded/parquet
/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [shaded.parquet.org.slf4j.helpers.NOPLoggerFactory]
17/08/28 16:09:30 WARN SuspiciousConnectsAnalysis: Total records discarded due to NULL values in key fields: 1156 . Please go to /user/spotuser
/flow/scored_results/20170804/scores/invalid_records for more details.

real    21m30.030s
user    2m26.056s
sys     0m7.804s
spotuser@cloudera-host-1:~/spot-ml$
```

Figure 4.7: Example of the execution of the anomaly detection script on netflow data

The output of the algorithm is a .csv file that contains the most suspicious records identified by the engine. In our example, this file can be reached from the HDFS by executing:

```
$ hdfs dfs -cat
/user/spotuser/flow/scored_results/20170705/scores/flow_results.csv
```

An example of the aforementioned file, with each record's suspicious score displayed at the last column is shown in Figure 4.8:

1	2016-01-27 23:48:43	2016	1	27	23	48	43	0.0	196.183.37.75	73.128.152.165	7150	1024	TCP	1	56	0	0	5.689344149379704E-9
2	2016-01-27 21:41:59	2016	1	27	21	41	59	0.164	73.128.153.229	196.183.37.75	1024	80	TCP	36	1872	0	0	1.55897504610095E-8
3	2016-01-27 21:41:59	2016	1	27	21	41	59	0.164	73.128.153.229	196.183.37.75	1024	80	TCP	36	1872	0	0	1.55897504610095E-8
4	2016-01-27 23:48:43	2016	1	27	23	48	43	0.004	73.128.152.165	196.183.37.75	1024	7150	TCP	2	108	0	0	1.6223040047334708E-8
5	2016-01-27 23:48:43	2016	1	27	23	48	43	0.004	73.128.152.165	196.183.37.75	1024	7150	TCP	2	108	0	0	1.6223040047334708E-8
6	2016-01-27 22:51:15	2016	1	27	22	51	15	0.652	73.128.153.119	196.183.37.75	19077	443	TCP	39	3352	0	0	1.6305524993092656E-8
7	2016-01-27 22:51:15	2016	1	27	22	51	15	0.652	73.128.153.119	196.183.37.75	19077	443	TCP	39	3352	0	0	1.6305524993092656E-8
8	2016-01-27 23:49:27	2016	1	27	23	49	27	0.0	73.128.152.165	196.183.37.75	1024	7150	TCP	2	104	0	0	1.631059335358762E-8
9	2016-01-27 23:49:27	2016	1	27	23	49	27	0.0	73.128.152.165	196.183.37.75	1024	7150	TCP	2	104	0	0	1.631059335358762E-8
10	2016-01-27 21:04:10	2016	1	27	21	4	10	0.0	73.128.153.8	196.183.37.75	1024	80	TCP	1	1498	0	0	1.6321198771383162E-8
11	2016-01-27 21:04:11	2016	1	27	21	4	11	0.0	73.128.153.8	196.183.37.75	1024	80	TCP	1	1498	0	0	1.6321198771383162E-8
12	2016-01-27 20:43:59	2016	1	27	20	43	59	0.116	196.183.37.75	73.128.153.131	80	1024	TCP	34	46720	0	0	1.6332644553491205E-8
13	2016-01-27 20:43:59	2016	1	27	20	43	59	0.116	196.183.37.75	73.128.153.131	80	1024	TCP	34	46720	0	0	1.6332644553491205E-8
14	2016-01-27 23:49:17	2016	1	27	23	49	17	0.0	196.183.37.75	73.128.152.165	7150	1024	TCP	1	802	0	0	1.6342152693659736E-8
15	2016-01-27 23:49:17	2016	1	27	23	49	17	0.0	196.183.37.75	73.128.152.165	7150	1024	TCP	1	802	0	0	1.6342152693659736E-8
16	2016-01-27 20:06:23	2016	1	27	20	6	23	18.88	67.135.252.253	224.192.15.223	68	67	UDP	5	1640	0	0	1.6648276957713935E-8
17	2016-01-27 19:22:38	2016	1	27	19	22	38	0.252	196.183.37.74	73.128.153.43	139	1024	TCP	35	17146	0	0	1.8719059788319058E-8
18	2016-01-27 19:22:38	2016	1	27	19	22	38	0.252	196.183.37.74	73.128.153.43	139	1024	TCP	35	17146	0	0	1.8719059788319058E-8
19	2016-01-27 23:49:42	2016	1	27	23	49	42	0.024	73.128.152.165	196.183.37.75	1024	7150	TCP	3	156	0	0	1.917537338756392E-8
20	2016-01-27 23:49:42	2016	1	27	23	49	42	0.024	73.128.152.165	196.183.37.75	1024	7150	TCP	3	156	0	0	1.917537338756392E-8
21	2016-01-27 23:49:07	2016	1	27	23	49	7	0.068	73.128.152.165	196.183.37.75	1024	7150	TCP	12	9624	0	0	2.5781438280307748E-8
22	2016-01-27 23:48:53	2016	1	27	23	48	53	0.0	73.128.152.165	196.183.37.75	1024	7150	TCP	11	8822	0	0	2.5781438280307748E-8
23	2016-01-27 23:49:07	2016	1	27	23	49	7	0.068	73.128.152.165	196.183.37.75	1024	7150	TCP	12	9624	0	0	2.5781438280307748E-8
24	2016-01-27 19:22:12	2016	1	27	19	22	12	0.004	196.183.37.74	73.128.154.75	80	14012	TCP	5	2404	0	0	3.176535566359507E-8
25	2016-01-27 22:12:10	2016	1	27	22	12	10	0.0	73.128.153.229	196.183.37.75	60906	1024	TCP	2	1604	0	0	3.262633138627752E-8
26	2016-01-27 22:12:10	2016	1	27	22	12	10	0.0	73.128.153.229	196.183.37.75	60906	1024	TCP	2	1604	0	0	3.262633138627752E-8
27	2016-01-27 20:29:28	2016	1	27	20	29	28	0.0	196.183.37.75	73.128.153.166	1024	25535	TCP	2	1604	0	0	3.262633138627752E-8
28	2016-01-27 20:29:28	2016	1	27	20	29	28	0.0	196.183.37.75	73.128.153.166	1024	25535	TCP	2	1604	0	0	3.262633138627752E-8
29	2016-01-27 19:44:22	2016	1	27	19	44	22	0.016	73.128.154.158	196.183.37.75	1024	14913	TCP	4	212	0	0	3.340111612118506E-8
30	2016-01-27 19:44:22	2016	1	27	19	44	22	0.016	73.128.154.158	196.183.37.75	1024	14913	TCP	4	212	0	0	3.340111612118506E-8
31	2016-01-27 19:46:43	2016	1	27	19	46	43	0.048	73.128.153.18	196.183.37.74	1024	52414	TCP	11	852	0	0	3.66577375292205E-8
32	2016-01-27 19:46:43	2016	1	27	19	46	43	0.048	73.128.153.18	196.183.37.74	1024	52414	TCP	11	852	0	0	3.66577375292205E-8
33	2016-01-27 19:44:22	2016	1	27	19	44	22	0.016	196.183.37.75	73.128.154.158	14913	1024	TCP	6	4412	0	0	4.036108579486876E-8
34	2016-01-27 19:44:22	2016	1	27	19	44	22	0.016	196.183.37.75	73.128.154.158	14913	1024	TCP	6	4412	0	0	4.036108579486876E-8
35	2016-01-27 21:29:22	2016	1	27	21	29	22	0.0	196.183.37.74	73.128.154.196	33346	1024	TCP	1	60	0	0	4.132951244026219E-8
36	2016-01-27 21:29:22	2016	1	27	21	29	22	0.0	196.183.37.74	73.128.154.196	33346	1024	TCP	1	60	0	0	4.132951244026219E-8
37	2016-01-27 23:48:44	2016	1	27	23	48	44	0.128	196.183.37.75	73.128.152.165	7150	1024	TCP	28	1456	0	0	4.152405310893494E-8

Figure 4.8: Example of the anomaly detection .csv file generated by the machine-learning node

4.4.3 Operational analytics and visualisation for netflow data

After the conclusion of the anomaly detection phase, the user can access the browser-based analyst view for Suspicious Connects and select the date that he wants to review, exploiting a number of monitoring screens. To enable the visualisation and operational analytics functionalities of our performed analysis, we must first go to the `~/.../spot-oa/oa` level of the spot-oa node and execute the following command:

```
$ python2.7 start_oa.py -d 2010705 -t flow -l 3000
```

The above customisable command will process the top 3000 records of the flow_results.csv file that was created by the machine-learning node, in order to create a detailed interactive network view of the analysis. The generated interface allows the user to inspect detected threats in the form of datatables and dendrograms, search for specific records, set his own scoring (in case false positives/negatives have occurred), view a summary of the ingested data, get geolocation information of the suspicious records etc. Example views of the GUI are shown below (Figures 4.9, 4.10, 4.11):

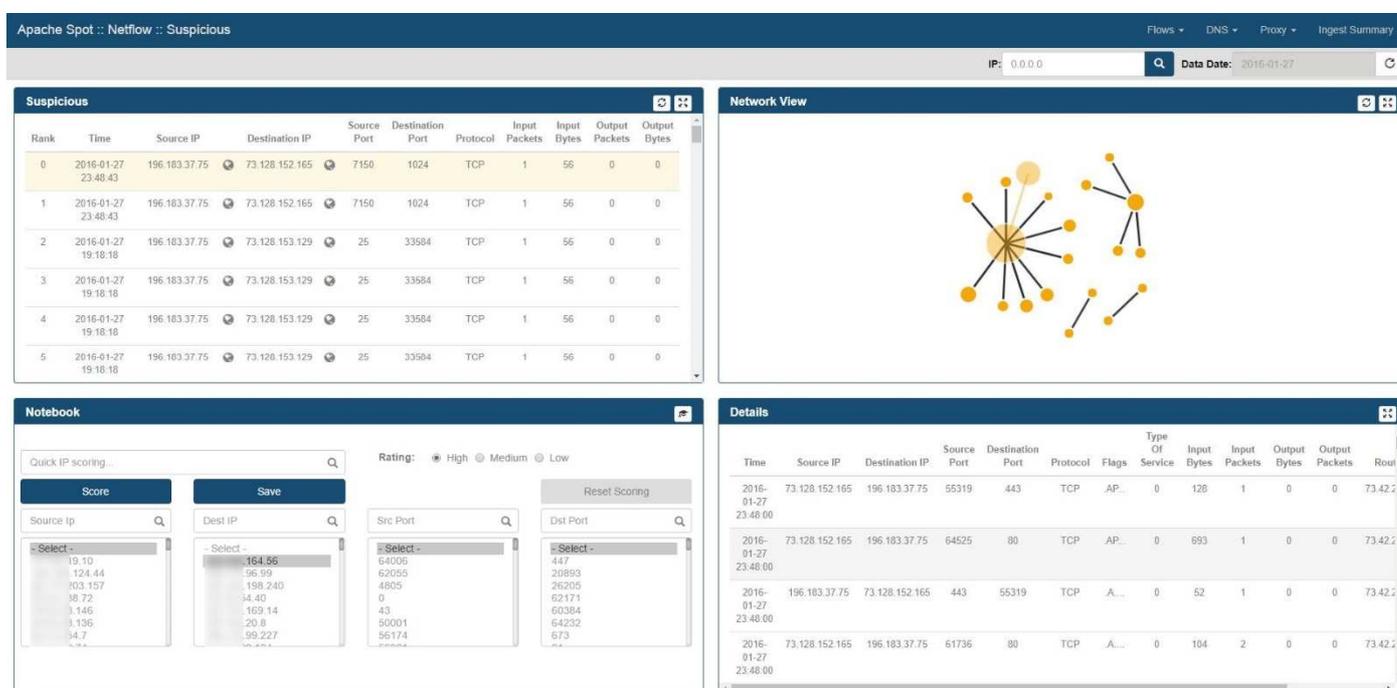


Figure 4.9: Suspicious analysis view of the Spot browser-based GUI. Top-left: The most suspicious connections are being shown at the in a descending order. Top-right: The connections are depicted in the form of dendrograms for better visibility. Bottom left: The ipython scoring notebook that allows manual score assignment to each connection. Bottom right: The set of records that comprise the selected (orange) connection between a source and a destination IP of the top-left window.

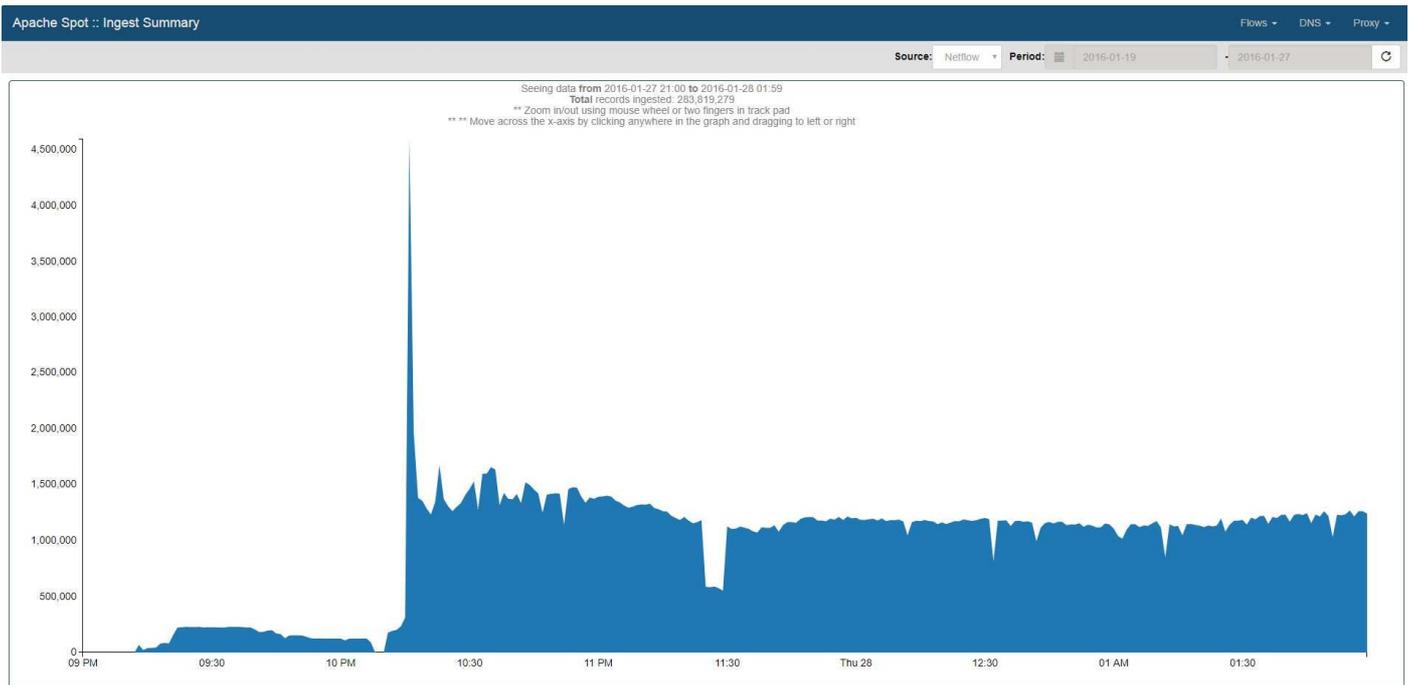


Figure 4.10: Ingest summary view of the Spot browser-based GUI for netflow ingestion

This concludes an example utilisation of the cybersecurity engine for netflow traffic logs that can be easily related to DNS and proxy datasets as well. In the next chapter, the engine will be tested with publicly available and captured datasets, exploiting all available pipelines, in order to infer useful knowledge regarding its effectiveness.

Chapter 5

Anomaly detection testing and findings

5.1 Description of test cases

The previous chapter introduced the reader to the implementation and utilisation details of a typical cybersecurity engine based on the exploitation of Big Data and machine-learning methods. The main components that comprise the engine were described and an example analysis of netflow traffic logs was outlined, in order to better comprehend their functionalities. In the present chapter, the Cognitive Data Analysis engine is submitted to different test cases, that will help us render useful conclusions regarding the engine's anomaly detection effectiveness, by discussing performance indicators such as detection rate, ranking of suspicious records, appearance of false positives/negatives, processing time, etc. By evaluating the results, it will become apparent that the presented cybersecurity implementation has its strong and weak points depending on the test case, as it doesn't produce optimal results in every trial. This comes in compliance with a very basic property that characterises machine-learning methods, which is widely known as the "No Free Lunch Theorem" (Wolpert, 1995). This theorem (mainly used for supervised methods but applicable here as well) simply states that there is no one model that works best for every problem, meaning that the assumptions of a great model for one problem may not hold for another, so it is common in machine learning to try multiple models and find one that works best for a particular problem. In other words, there is no standard machine-learning algorithm which can be applicable for every problem and get the best results.

The test cases to be presented in the next sections involve the analysis of two main dataset types:

- publicly available datasets that come with predefined malicious network behaviour, including a large number of attacks
- captured datasets from a testbed's local network, where attacks were performed by using relevant penetration tools

Each test exploits a different data pipeline (netflow, DNS, proxy) while the ingested data may include one or more types of network threats, depending on the case.

5.2 Analysis on publicly available datasets

The data exploited in this section has been acquired by a repository maintained by the Apache Spot community¹³. This repository includes a number of compressed network traffic files of different types with several performed attacks and is meant for the testing and showcasing of the engine's capabilities. More specifically, at the time of writing this thesis the network files that were available (and therefore used for our test cases) included:

- ~2.3GB of netflow files (.nfcapd format)
- ~20GB of DNS files (.pcap format)
- ~300MB of proxy files (bluecoat .csv format)

The above datasets were ingested by our testbed's engine and the results are documented in the following subsections.

5.2.1 Test on netflow data

The first test was conducted on a single day's netflow traffic dataset (27/1/2016) that collectively included more than 52 million records. The dataset included a port scan attempt, common example of suspicious behaviour as network ports are the entry points

¹³ Public network traffic logs, offered by the Spot community:

<https://issues.apache.org/jira/browse/SPOT-135?jql=project%20%3D%20SPOT>

to a connected machine and malicious clients try to exploit vulnerabilities so they can gain access to sensitive data or remotely execute malicious code. Port scanning is usually done in the initial phase of a penetration test in order to discover all network entry points into the target system. In our case, scanning was limited to a number of common TCP ports, such as: 25 (SMTP), 53 (DNS), 80 (HTTP), 443(SSL/TLS) etc. The ingested data also included a connection to the application port 1024, which is related to backdoor activity as it is used by many remote administration tools (RATs).

Apache Spot identified both threats, by assigning an extremely small probability score to the aforementioned connections, resulting in them ranking at the top of the suspicious analysis list (Figure 5.1). More specifically, the record that pointed to a potential remote access attempt (port 1024 backdoor) was ranked first with a probability score of $8.43e-9$, while the records related to port scanning were following with a probability score ranging from $1.21e-8$ to $4.98e-8$. It should be noted that the default probability threshold below which the engine labels a record as suspicious is $1e-6$.

The duration of the analysis was 25 minutes and the overall size of the decompressed .nfcapd files that were ingested by the engine was 4GBs.

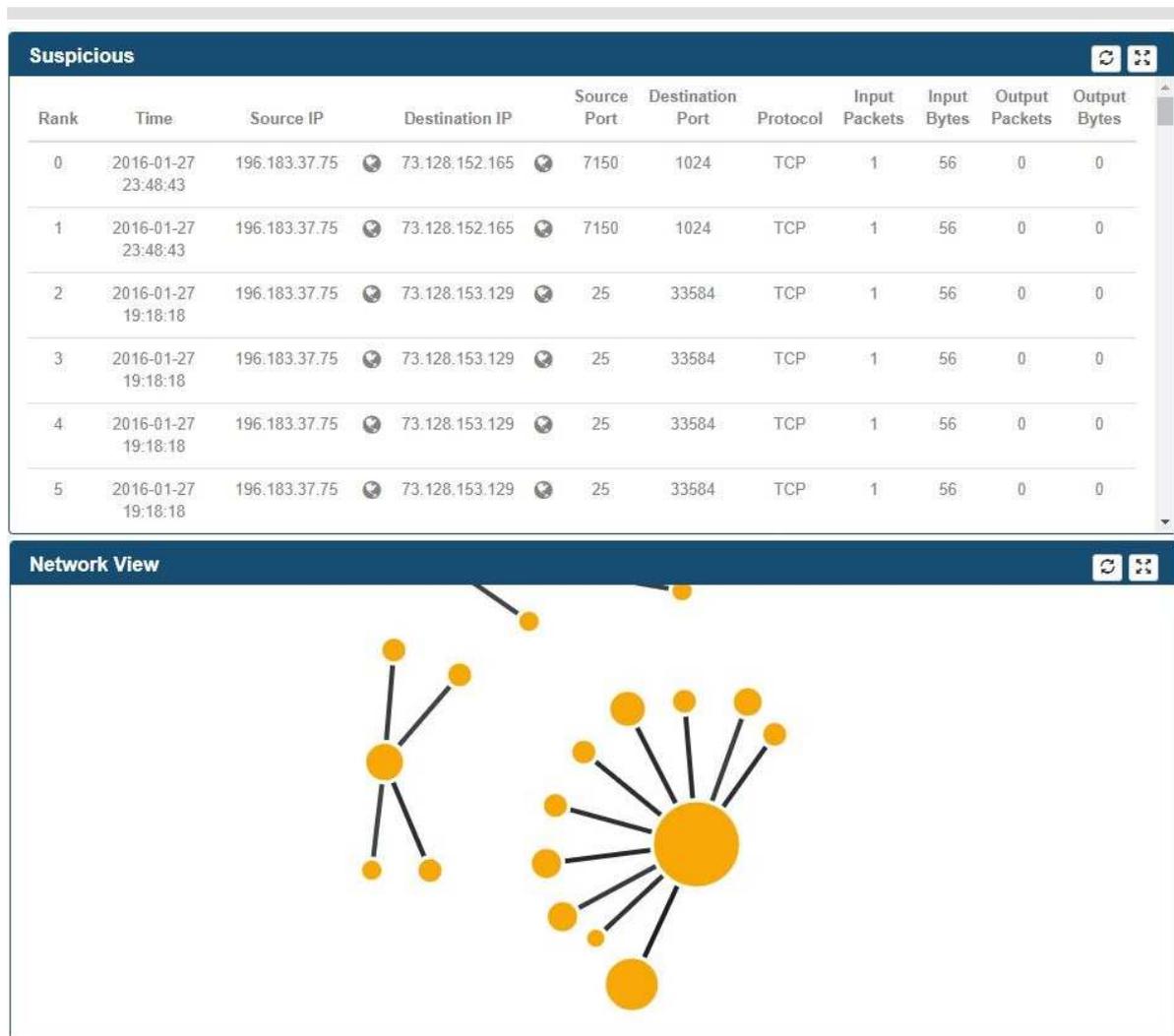


Figure 5.1: View of the analysis results on netflow data

5.2.2 Test on DNS data

Using the publicly available DNS datasets, a test was executed for the available .pcap files of a single day (04/01/2017), which collectively included more than 11 million logs. In general, by looking at .pcap files we are able to identify suspicious queries that might lead to a number of different DNS attacks. A DNS attack is an exploit in which an attacker takes advantage of vulnerabilities in the domain name system. Given that DNS is a protocol that translates a user-friendly domain name, into the computer-friendly IP address, when a

user types a domain name into a browser this has to be translated to an IP address. A program in the client's operating system called DNS resolver first checks its own local cache to see if it already has the IP address and if this isn't the case, it queries a DNS server to see if it knows the correct IP address for the typed domain name. DNS servers are recursive, which means that they can query each other to either find another DNS server that knows the correct IP address or find the authoritative DNS server that stores the canonical mapping of the domain name to its IP address. As soon as the resolver locates the IP address, it returns it to the requesting program and caches the address for future use. This communication back and forth between clients and servers is vulnerable and the types of DNS attacks in use today that can take advantage of it are numerous and quite complex. The most common threats, some of which are included in this dataset are:

- DNS amplification attacks, where the attacker takes advantage of a DNS server that permits recursive lookups and uses recursion to spread his attack to other DNS servers
- Cache poisoning, where the attacker corrupts a DNS server by replacing a legitimate IP address in the server's cache with that of another rogue address in order to redirect traffic to a malicious website, collect information or initiate another attack.
- DNS tunnelling, which is a common way to establish connections with remote systems, get remote access or exfiltrate data from users, by encoding the data/response to the server in the hostname that is being requested.

In our case, the anomaly detection procedure managed to identify a great number of queries from the same client IP to various server IPs with unusual behaviour (Figure 5.2). For example, while the same query (domain name of a Russian website) was requested, every response was a different IP address, probably pointing to a cache poisoning situation. Moreover, this particular IP had generated a big amount of queries for suspicious domain names with varying query size and during abnormal hours (after midnight), most likely signifying that a malware/rootkit was deployed in the client's machine and was extracting

information via a DNS tunnelling attempt. The fluctuating query size, the abnormal hours of the requests and the different query responses triggered the anomaly detection procedure, thus these connections were marked as top suspicious with scores varying from $8.3e-9$ to $8.5e-9$.

The duration of the analysis was 80 minutes and the overall size of the decompressed .pcap files that were ingested by the engine was 64GBs.



Figure 5.2: View of the analysis results on DNS data

5.2.3 Test on proxy data

Using the available proxy dataset for a specific date (28/6/2016), the machine learning algorithm processed more than 800 thousand bluecoat logs. The word creation procedure for proxy data is based on factors including proxy host, connection time, request method (POST, GET, HEAD, CONNECT, etc.), the Uniform Resource Identifier (URI) which is the string of characters used to identify a resource, content type, user agent (browser), response code etc. All the ingested logs are also compared with an online “safe list” of top domains that helps reduce false-positive alerts as part of a “pre-LDA” procedure.

The ingested dataset included suspicious after hours connections to unknown financial websites, redirection requests probably linking to botnet activity and data exfiltration, high upload/download activity to/from suspicious domains that could signify data leakage from installed malware etc. The anomaly detection procedure successfully identified hundreds of connections as potentially malicious by assigning suspicious scores varying from $6.1e-7$ to $5.8e-6$ (Figure 5.3). For better threat visibility, the engine offers the ability to correlate the each proxy record’s reputation with a proprietary threat intelligence interface like McAfee’s GTI¹⁴ or Facebook’s Threat Exchange¹⁵. These technologies use cloud querying to render a response in the form of a reputation score or categorization information for each connection. However, due to the requirement of an active licence, they were not used in our tests.

The duration of the analysis was 16 minutes and the overall size of the decompressed .log files that were ingested by the engine was 371MBs.

14 McAfee Global Threat Intelligence (GTI):

<https://kc.mcafee.com/corporate/index?page=content&id=KB70130> Accessed July 2017

15 Facebook Threat Exchange: <https://developers.facebook.com/products/threat-exchange>

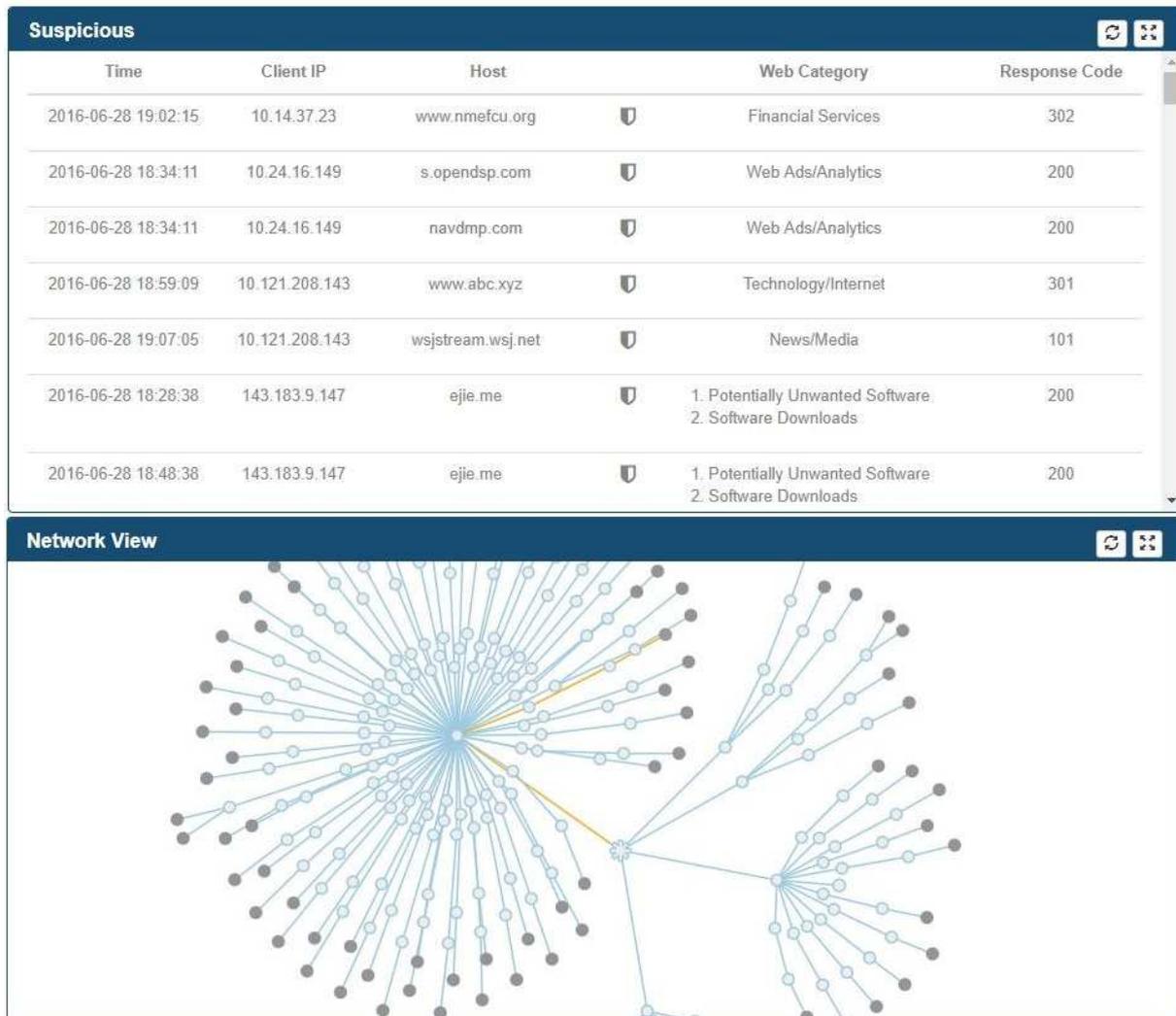


Figure 5.3: View of the analysis results on proxy data

5.3 Analysis on captured data

The previous tests performed on sample datasets have clearly shown that the Apache Spot engine is capable of detecting a wide range of network threats. However, these datasets included malicious network behaviour that was not documented, meaning that we could not beforehand know anything about the generated threats until the anomaly detection procedure was executed. Naturally, even after the execution of the machine-learning algorithm, there was no way to specify if all the generated threats were detected or

not. Thus, additional tests had to be made, this time by examining captured traffic from local networks enriched with network attacks performed by penetration testing tools. Although generating enough normal traffic to properly feed the machine-learning algorithm can be difficult in such small networks, it was the only way to properly test its capabilities in specific attack use cases.

5.3.1 DoS detection using the netflow pipeline

Using our testbed environment, we first performed a UDP DoS flood to a website supported by a server in our network, in order to test Spot's flow detection. A UDP flood is a type of Denial of Service (DoS) attack in which the attacker overwhelms random ports on the targeted host with IP packets containing UDP datagrams. The receiving host checks for applications associated with these datagrams and, finding none, ends back a “Destination Unreachable” ICMP packet. As more and more UDP packets are received and answered, the system becomes overwhelmed and unresponsive to other clients. In the framework of a UDP flood attack, the attacker may also spoof the IP address of the packets, both to make sure that the return ICMP packets don't reach their host, and to anonymize the attack. In our test, we used BoNeSi¹⁶, an open-source DDoS Botnet Simulator that can simulate botnet traffic in a testbed environment. It is designed to study the effect of DDoS attacks and can generate UDP/TCP/ICMP packets from/to a number of IPs with a predefined intensity. The detailed specifications of the test are shown in Table 5.1:

Date:	26/7/2017
Traffic:	Bonesi + normal (total=5434188 records)
Attack:	UDP flood DoS (1 source IP)

¹⁶ BoNeSi DDoS Botnet Simulator: <https://github.com/Markus-Go/bonesi> Accessed July 2017

Package generation rate:	1000packets/sec
Attack duration:	1min
Source IP:	10.101.30.58
Destination IP:	143.233.227.71:80
Spot rank of the attack:	2927 th / 5434188

Table 5.1: DoS attack specifications

The UDP flood records were ranked much lower than the top records of the flow_results.csv list, while the records being shown as "top suspicious" were mostly false-positives as they consisted of random ICMP and other packets (Figure 5.4). The results led to the conclusion that the anomaly detection algorithm was unable to identify the flow of UDP packets as suspicious with the current traffic mixture.



Figure 5.4: View of the DoS detection results

A second test was conducted using BoNeSi, leveraging this time the distributed attack capabilities of the tool to simulate a DDoS. A Distributed Denial of Service (DDoS) attack is a subclass of denial of service (DoS) attacks. It involves multiple connected online devices, collectively known as a botnet, which are used to overwhelm a target website with fake traffic. In our case, we used the Apache Spot engine to detect a simulated UDP flood with 10 source IPs, attacking the same website as above. The specifications of the attack are given below (Table 5.2):

Date:	2/8/2017
Traffic:	Bonesi + normal (total= 8051816records)
Attack:	UDP flood DDoS (10 source IPs)
Package generation rate:	1000packets/sec per IP
Attack duration:	3min
Source IP:	10.101.30.61-70
Destination IP:	143.233.227.71:80
Spot rank of the attack	9471 st / 8051816

Table 5.2: Distributed DoS attack specifications

Again, the engine was not capable of detecting the DDoS attack on the suspicious connections list (Figure 5.5). The “top suspicious” records were random connections (false-positives), while the actual attack records were ranked very low, confirming the validity of the previous test’s conclusion.

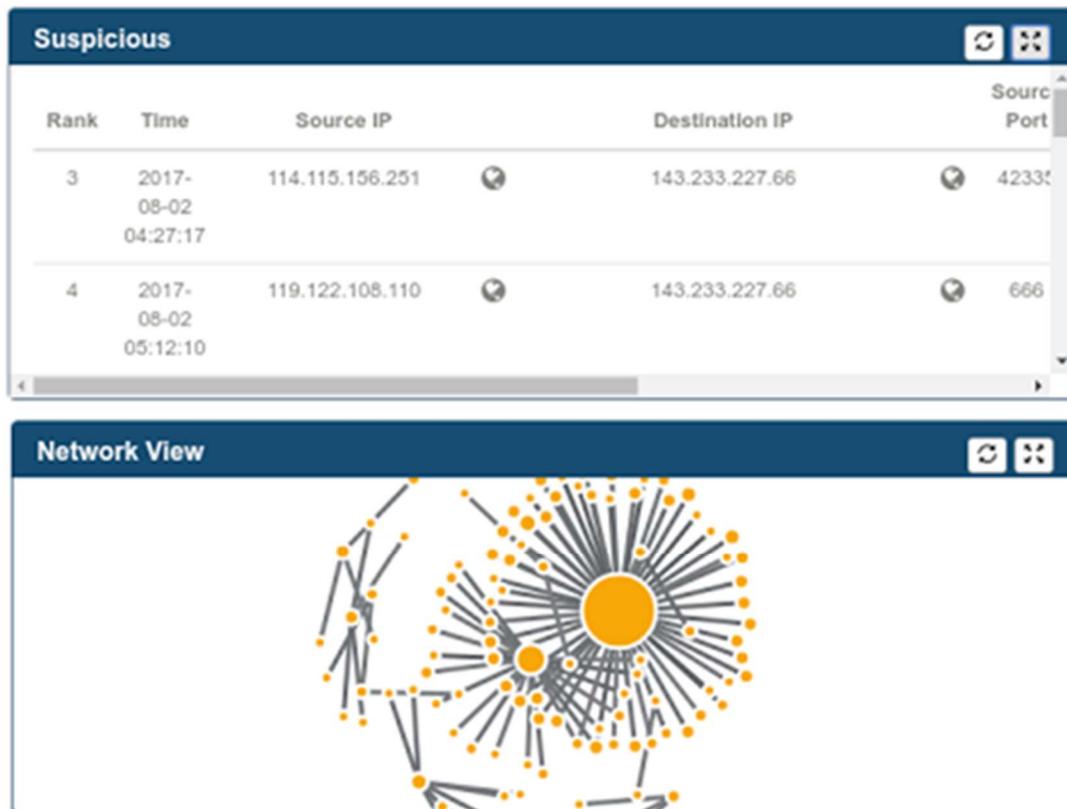


Figure 5.5: View of the Distributed DoS detection results

5.3.2 Anomaly detection in enterprise-level network traffic

The results of the two previous simulations of §5.3.1 indicated that the machine-learning process has a difficulty in characterising malicious behaviour (UDP flood) as an anomaly of normal network traffic. However, due to the fact that the tests were conducted in a small network testbed, it is possible that the traffic mixture was not optimal for anomaly detection. More specifically, because of the low activity of the monitored network, it is possible that the ingestion process didn't manage to accumulate enough normal network traffic in order for the LDA algorithm to correctly create the generic topics that describe benign behaviour. Since the engine can only process a day's network traffic, it was difficult

to further populate the network logs with normal activity, in order to test this possibility. Thus, a larger, more active network was needed in order to verify whether the low ranking of the DDoS records was due to bad traffic mixture or due to the algorithm's inherent difficulty to characterize these logs as outliers.

The following test was performed after request to the Apache Spot community in Intel's datacentre at Guadalajara Design Centre (GDC), Mexico. The utilised network consisted of a 10 Gb/sec Ethernet interface with a 40-50% load on normal situations (not while performing the attacks). Instead of simulating a single attack, the BreakingPoint¹⁷ solution was used. This tool can generate real-world legitimate traffic and supports more than 37000 threats, including distributed denial of service (DDoS), exploits, malware, and fuzzing attacks. The test was conducted on 31/7/2017 and its overall duration was 6 hours. The following table (Table 5.3) includes the top 100 attacks detected by Spot from the overall 6918 attacks that were simulated. Some of the fields (date-time, source/destination ports and original string) have been removed from the original list for better visibility and the attacks are sorted by Spot rank (4th column) that illustrates their place in the suspicious analysis of the netflow traffic.

Source IP	Dest. IP	Susp. Score	Spot rank	Protocol	Attack name
122.52.235.65	25.0.19.29	8.83E-09	39	TCP	Novell iPrint Server natural languages Buffer Overflow
122.52.235.65	25.0.19.29	8.83E-09	39	TCP	Novell iPrint Server natural languages Buffer Overflow
12.166.7.88	25.0.19.51	1.03E-08	55	TCP	Freefloat FTP Stack Buffer Overrun
12.166.7.88	25.0.19.51	1.03E-08	55	TCP	Freefloat FTP Stack Buffer Overrun
122.52.236.28	25.0.18.211	1.11E-08	59	UDP	Linux Kernel UDP UFO Denial of Service
122.52.236.28	25.0.18.211	1.11E-08	59	UDP	Linux Kernel UDP UFO Denial of Service

¹⁷ BreakingPoint Network security testing platform: <https://www.ixiacom.com/products/network-security-testing-breakingpoint> Accessed July 2017

122.52.236.205	25.0.18.98	1.16E-08	61	TCP	OpenSSL Warnings and Finished after Client Hello Null Pointer Dereference
122.52.236.205	25.0.18.98	1.16E-08	61	TCP	OpenSSL Warnings and Finished after Client Hello Null Pointer Dereference
122.52.250.1	25.0.18.244	1.25E-08	71	TCP	Samba v3 Malformed AndX Request DoS
122.52.250.1	25.0.18.244	1.25E-08	71	TCP	Samba v3 Malformed AndX Request DoS
12.166.6.183	25.0.18.212	1.38E-08	149	TCP	Doms Httpd Denial of Service
12.166.6.183	25.0.18.212	1.38E-08	149	TCP	Doms Httpd Denial of Service
80.66.17.62	25.0.19.72	1.39E-08	160	UDP	RSYSLOG PRI Value Parsing Denial of Service
80.66.17.62	25.0.19.72	1.39E-08	160	UDP	RSYSLOG PRI Value Parsing Denial of Service
122.52.235.35	25.0.17.236	1.70E-08	314	TCP	Samba v3 Malformed AndX Request DoS
122.52.235.35	25.0.17.236	1.70E-08	314	TCP	Samba v3 Malformed AndX Request DoS
122.52.241.47	25.0.19.75	1.73E-08	319	TCP	Microsoft Vista Negotiate Protocol Dialects DoS
122.52.241.47	25.0.19.75	1.73E-08	319	TCP	Microsoft Vista Negotiate Protocol Dialects DoS
12.166.8.34	25.0.18.210	1.80E-08	365	TCP	Novell iPrint LPD Buffer Overflow
12.166.8.34	25.0.18.210	1.80E-08	365	TCP	Novell iPrint LPD Buffer Overflow
122.52.252.246	25.0.18.166	1.85E-08	386	TCP	Freefloat FTP Stack Buffer Overrun
122.52.252.246	25.0.18.166	1.85E-08	386	TCP	Freefloat FTP Stack Buffer Overrun
122.52.236.205	25.0.18.45	1.92E-08	419	TCP	Windows GDI Malformed Image Denial of Service (POP3)
122.52.236.205	25.0.18.45	1.92E-08	419	TCP	Windows GDI Malformed Image Denial of Service (POP3)
122.52.240.108	25.0.18.128	1.98E-08	449	TCP	Microsoft Windows 2000 microsoft-ds DoS TCP
122.52.240.108	25.0.18.128	1.98E-08	449	TCP	Microsoft Windows 2000 microsoft-ds DoS TCP
122.52.250.124	25.0.19.17	1.99E-08	457	TCP	Adobe Flash 10 Corrupted SWF File
122.52.250.124	25.0.19.17	1.99E-08	457	TCP	Adobe Flash 10 Corrupted SWF File
122.52.244.85	25.0.19.25	1.99E-08	458	TCP	Internet Explorer RDS.DataControl.URL DoS
122.52.244.85	25.0.19.25	1.99E-08	458	TCP	Internet Explorer RDS.DataControl.URL DoS
122.52.240.53	25.0.18.238	2.01E-08	465	UDP	Windows Media Player RTSP Double Free
122.52.240.53	25.0.18.238	2.01E-08	465	UDP	Windows Media Player RTSP Double Free
80.66.18.32	25.0.17.136	2.06E-08	475	TCP	RealNetworks Helix Server Denial of Service
80.66.18.32	25.0.17.136	2.06E-08	475	TCP	RealNetworks Helix Server Denial of Service
122.52.233.168	25.0.18.43	2.06E-08	480	TCP	Novell GWIA IMAP Stack Overflow
122.52.233.168	25.0.18.43	2.06E-08	480	TCP	Novell GWIA IMAP Stack Overflow

80.66.18.133	25.0.17.232	2.40E-08	582	TCP	Doms Httpd Denial of Service
80.66.18.133	25.0.17.232	2.40E-08	582	TCP	Doms Httpd Denial of Service
12.166.7.106	25.0.17.112	2.40E-08	583	UDP	Microsoft SMB Browser Election Pool Corruption
12.166.7.106	25.0.17.112	2.40E-08	583	UDP	Microsoft SMB Browser Election Pool Corruption
213.184.19.65	25.0.18.152	2.40E-08	584	UDP	Microsoft SMB Browser Election Pool Corruption
213.184.19.65	25.0.18.152	2.40E-08	584	UDP	Microsoft SMB Browser Election Pool Corruption
122.52.245.171	25.0.18.31	2.41E-08	590	UDP	Windows NT Server 4.0 Hostname Buffer Overflow
122.52.245.171	25.0.18.31	2.41E-08	590	UDP	Windows NT Server 4.0 Hostname Buffer Overflow
122.52.255.69	25.0.18.31	2.41E-08	591	UDP	Windows NT Server 4.0 Hostname Buffer Overflow
122.52.255.69	25.0.18.31	2.41E-08	591	UDP	Windows NT Server 4.0 Hostname Buffer Overflow
213.184.22.175	25.0.17.202	2.66E-08	650	TCP	Novell iPrint LPD Buffer Overflow
213.184.22.175	25.0.17.202	2.66E-08	650	TCP	Novell iPrint LPD Buffer Overflow
213.184.21.177	25.0.17.120	3.07E-08	712	TCP	OpenSSL Anonymous ECDH Handshake NULL Pointer Dereference
213.184.21.177	25.0.17.120	3.07E-08	712	TCP	OpenSSL Anonymous ECDH Handshake NULL Pointer Dereference
80.66.16.33	25.0.17.132	3.07E-08	713	UDP	Solaris DHCP Malformed BOOTP Packet Denial of Service
80.66.16.33	25.0.17.132	3.07E-08	713	UDP	Solaris DHCP Malformed BOOTP Packet Denial of Service
80.66.15.225	25.0.19.70	3.07E-08	714	UDP	Solarwinds TFTP Server Denial of Service
80.66.15.225	25.0.19.70	3.07E-08	714	UDP	Solarwinds TFTP Server Denial of Service
213.184.19.116	25.0.18.74	3.15E-08	715	TCP	RealNetworks Helix Server Denial of Service
213.184.19.116	25.0.18.74	3.15E-08	715	TCP	RealNetworks Helix Server Denial of Service
12.166.7.155	25.0.18.93	3.22E-08	722	TCP	MIT Kerberos rcvauth Invalid Memory Access Denial of Service
12.166.7.155	25.0.18.93	3.22E-08	722	TCP	MIT Kerberos rcvauth Invalid Memory Access Denial of Service
80.66.16.195	25.0.18.162	3.30E-08	747	UDP	RSYSLOG PRI Value Parsing Denial of Service
80.66.16.195	25.0.18.162	3.30E-08	747	UDP	RSYSLOG PRI Value Parsing Denial of Service
80.66.16.147	25.0.18.73	3.31E-08	758	TCP	Microsoft RPCSS DCOM Interface Denial of Service
80.66.16.147	25.0.18.73	3.31E-08	758	TCP	Microsoft RPCSS DCOM Interface Denial of Service

213.184.22.197	25.0.18.78	3.31E-08	759	TCP	Microsoft RPCSS DCOM Interface Denial of Service
213.184.22.197	25.0.18.78	3.31E-08	759	TCP	Microsoft RPCSS DCOM Interface Denial of Service
122.52.251.121	25.0.18.209	3.36E-08	772	TCP	Microsoft IIS 7.5 FTPSVC Telnet IAC DoS
122.52.251.121	25.0.18.209	3.36E-08	772	TCP	Microsoft IIS 7.5 FTPSVC Telnet IAC DoS
213.184.20.137	25.0.19.76	3.41E-08	783	IP	Cisco SWIPE Denial of Service
213.184.20.137	25.0.19.76	3.41E-08	783	IP	Cisco SWIPE Denial of Service
213.184.19.75	25.0.19.126	3.51E-08	853	UDP	EMC NetWorker Denial of Service
213.184.19.75	25.0.19.126	3.51E-08	853	UDP	EMC NetWorker Denial of Service
213.184.20.92	25.0.19.2	3.51E-08	854	UDP	EMC NetWorker Denial of Service
213.184.20.92	25.0.19.2	3.51E-08	854	UDP	EMC NetWorker Denial of Service
213.184.20.172	25.0.19.65	3.66E-08	941	UDP	Cisco ASA IKE Fragment Length Negative Copy Denial of Service
213.184.20.172	25.0.19.65	3.66E-08	941	UDP	Cisco ASA IKE Fragment Length Negative Copy Denial of Service
12.166.7.202	25.0.17.210	3.69E-08	944	UDP	Digium Asterisk SIP INVITE Session Expiration Denial of Service
12.166.7.202	25.0.17.210	3.69E-08	944	UDP	Digium Asterisk SIP INVITE Session Expiration Denial of Service
213.184.19.40	25.0.17.152	3.71E-08	945	UDP	Solarwinds TFTP Server Denial of Service
213.184.19.40	25.0.17.152	3.71E-08	945	UDP	Solarwinds TFTP Server Denial of Service
213.184.21.46	25.0.17.245	3.74E-08	949	UDP	Linux Netfilter NAT SNMP DoS
213.184.21.46	25.0.17.245	3.74E-08	949	UDP	Linux Netfilter NAT SNMP DoS
122.52.246.50	25.0.18.198	3.75E-08	951	TCP	Microsoft IIS 7.5 FTPSVC Telnet IAC DoS
122.52.246.50	25.0.18.198	3.75E-08	951	TCP	Microsoft IIS 7.5 FTPSVC Telnet IAC DoS
12.166.7.147	25.0.19.14	3.77E-08	960	UDP	Linux Kernel UDP UFO Denial of Service
12.166.7.147	25.0.19.14	3.77E-08	960	UDP	Linux Kernel UDP UFO Denial of Service
213.184.19.32	25.0.19.23	3.94E-08	1007	TCP	MIT Kerberos rcvauth Invalid Memory Access Denial of Service
213.184.19.32	25.0.19.23	3.94E-08	1007	TCP	MIT Kerberos rcvauth Invalid Memory Access Denial of Service
122.52.227.171	25.0.18.227	3.95E-08	1008	TCP	ProFTP 2.9 Client Banner Buffer Overflow
122.52.227.171	25.0.18.227	3.95E-08	1008	TCP	ProFTP 2.9 Client Banner Buffer Overflow
122.52.232.153	25.0.18.8	4.10E-08	1014	TCP	Microsoft Word RTF Parsing Engine Stack Exhaustion (POP3)
122.52.232.153	25.0.18.8	4.10E-08	1014	TCP	Microsoft Word RTF Parsing Engine Stack Exhaustion (POP3)

122.52.237.9	25.0.18.108	4.19E-08	1016	TCP	Microsoft Word 2003 MSO Null Pointer Dereference DoS (POP3 Quoted Printable)
122.52.237.9	25.0.18.108	4.19E-08	1016	TCP	Microsoft Word 2003 MSO Null Pointer Dereference DoS (POP3 Quoted Printable)
80.66.19.134	25.0.18.140	4.21E-08	1017	TCP	Novell Netware Apple Filing Protocol DoS
80.66.19.134	25.0.18.140	4.21E-08	1017	TCP	Novell Netware Apple Filing Protocol DoS
122.52.243.27	25.0.18.233	4.24E-08	1018	UDP	Solaris DHCP Malformed BOOTP Packet Denial of Service
122.52.243.27	25.0.18.233	4.24E-08	1018	UDP	Solaris DHCP Malformed BOOTP Packet Denial of Service
12.166.7.180	25.0.19.83	4.42E-08	1021	UDP	Network Time Protocol CRYPTO_NAK Crash
12.166.7.180	25.0.19.83	4.42E-08	1021	UDP	Network Time Protocol CRYPTO_NAK Crash
213.184.22.219	25.0.19.110	4.51E-08	1023	UDP	Linux Netfilter NAT SNMP DoS
213.184.22.219	25.0.19.110	4.51E-08	1023	UDP	Linux Netfilter NAT SNMP DoS

Table 5.3: Ranking of attacks performed by BreakingPoint

As shown in the above table, several UDP DDoS attacks have been simulated and have been ranked as highly suspicious connections (Spot rank 59-960). These results confirm that the traffic mixture (normal:attack ratio) that is being ingested by the engine plays a major role to the effectiveness of the machine-learning algorithm. For instance, if the traffic related to an attack (e.g. UDP flood packets) can be compared in size with normal network traffic, it is possible that during the topic creation phase of the LDA implementation, these attack logs will form their own topic, meaning that any attack that can be correlated with it will be considered normal network behaviour.

One additional reason that would justify the higher ranking of the attacks performed in this subsection compared to the ones described in §5.3.1, is that in the previous case the attacker IPs were previously unused. Since the LDA implementation assigns a topic distribution per IP, it is only natural that a previously unused IP leveraged only for attack

purposes will be assigned a topic distribution that includes topics (which represent normal behaviour) with low probability for each one of them. This can blur the boundaries between benign and abnormal network behaviour, resulting in the categorisation of some attack as normal behaviour (false-negatives) while other non-suspicious records appear as malicious (false-positives).

In this chapter, the cybersecurity engine was tested against a large number of network threats, using both publicly available and captured datasets and exploiting all available data pipelines. The detection results from public datasets have shown that the engine has a good detection rate against unauthorised connection attempts (e.g. port probing) and suspicious querying (e.g. DNS tunnelling). The detection rate is worse for threats based on less unusual patterns such as a UDP DDoS, where the means of performing the attack is the generation of many packets of the same format. In all cases, the engine was having trouble dealing with false-positive alerts resulting in characterising normal records as suspicious. The performed tests also indicated a correlation between the traffic mixture and the effectiveness of the anomaly detection procedure. The extracted conclusions from the attack simulations as well as any other significant inference drawn in the course of this thesis are reported in the last chapter.

Chapter 6

Conclusions

In this chapter, we will discuss the conclusions of the work presented in this thesis and suggest the future work that could be done in the domain, in order to further exploit the potential of the utilised framework to advance the role of cognitive analytics in cybersecurity.

6.1 Concluding remarks

This research has explored the use of network traffic Big Data in conjunction with machine learning methods to achieve the detection and prevention of cybersecurity anomalies and threats. It was considered optimal to present the extracted conclusions per chapter, while recapitulating the main points that were raised.

In **Chapter 1**, the prevalence of Big Data in organizations' day-to-day activities was introduced. From the presentation of industry examples, it was made clear that its use is having a transformative effect not only on businesses but in our everyday life as well, affecting virtually all fields of human activity. New technologies are not only allowing us to collect Big Data in the most unimaginable ways, but also to constantly collect almost every byte of knowledge and activity on real time, all the time. This has inevitably generated new challenges regarding its acquisition, storage and exploitation that -when overcome- will definitely lead to the expansion of the scope of its applications, not just in the presented field of network analytics but in every applied field that requires the accumulation of information to produce knowledge. Big Data's tremendous rate of growth in complexity and volume may lead to the fact that what we call Big Data today to be just called data tomorrow.

In **Chapter 2**, the use of machine learning analytics in network security to leverage the information hidden in Big Data was presented. By comparing modern anomaly detection systems based on cognitive analytics with traditional signature-based approaches, it was made clear that -while suffering from their own weaknesses- the former surpass the latter in the detection of unknown and unprecedented threats that cannot be detected by rule-based solutions. The categorisation of machine-learning methods to supervised and unsupervised was thoroughly explained and some of the most common algorithms of each family were presented, along with cybersecurity-related implementations that help the reader to appreciate their value.

Chapters 3 and 4 were devoted to the presentation of a state-of-the-art cybersecurity engine, based on cognitive analytics. After describing the typical profile of an anomaly detection system based on modern bibliography, an overview of the developed Data Analysis and Remediation Engine (DARE) was given, proving that it follows the aforementioned profile. A detailed architectural view of the engine and its components was also provided, focusing on the phases of data acquisition and data analytics which constitute the core of the platform as well as the main scope of the thesis. The main technology that materialises the above phases, the Apache Spot platform, was presented and the algorithmic implementation of Latent Dirichlet Allocation (LDA) for network anomaly detection was explained in detail. Finally, an example use of the engine was demonstrated, depicting all important stages from the collection of data to the presentation of results.

In **Chapter 5**, the deployed anomaly detection engine was tested with publicly available datasets and captured network traffic, both containing logs of malicious activity. All tests performed on public datasets showcased the high detection capabilities of the

engine regarding a wide range of anomalies (e.g port scanning, dns tunnelling, botnet activity, data exfiltration etc.). However, the aforementioned tests were performed in “ideal conditions”, meaning that the engine was used in an enterprise-level network with high activity and state-of-the-art penetration testing and traffic simulation tools. The tests performed on traffic captured from our testbed’s network, indicated that the engine has a difficulty in detecting specific threats such as simulated Denial of Service attacks, due to a number of reasons that can be generalised to the overall detection capabilities of the engine and are presented below:

Firstly, it was observed that the detection procedure is highly dependent on the accumulated traffic mixture (normal: anomalous traffic ratio), as a result of the topic creation method that is used by the LDA algorithm. More specifically, since each network log is correlated with a topic distribution that represents normal (meaning most common) network behaviour, if the traffic profile consists of more anomalous traffic than benign, it is likely that the created topics will depict a false image of the network. As a result, the algorithm will distinguish normal traffic logs as suspicious outliers (false positives/ type I error) and at the same time will characterise suspicious behaviour as normal (false negatives / type II error). This is a common weakness of unsupervised learning methods since the algorithm is not trained with labelled data prior to deployment as in supervised learning. However, labelled datasets in network security only exist for a small number of known threats (which are constantly transforming, making detection more difficult) and even then it is difficult to acquire them.

Secondly, the need for a realistic topic distribution per IP for the LDA algorithm to successfully detect network anomalies was ascertained. This means that each IP must be assigned a unique topic distribution that describes its network behaviour, based on its activity. Thus, if a previously unused IP is used only for a DDoS attack, its topic mixture will not be easily correlated with any available topic, since the created topics only point to

normal behaviour and this IP was used solely for abnormal behaviour. As a result, the topic distribution of this IP will contain a mixture of normal topics, but the words (meaning the IP logs) associated with each topic will have very low probability, which is likely to lead to false negatives. The solution to this problem is to ensure that each IP included in the analysis has already been used for normal network activity.

Thirdly, although the detection results indicated the LDA algorithm's promising potential, the lack of correlation between detected anomalies and specific threats (e.g. DDoS) was clearly missing from the capabilities of the engine. In order to be able to provide a drastic defence response, users must first be aware of the specific threats they are dealing with. This implies that, in order to achieve integrated network security, the Apache Spot engine should not function in a standalone way but in collaboration with additional modules that will correlate the detected anomalies with specific threats/attacks, leveraging either rule-based or supervised machine learning methods. The development of such modules could also prove extremely helpful in the elimination of false-positive alerts.

Finally, it was made clear from the execution of the data ingestion and data analysis procedures that the extraction of useful information via batch processing requires significant amounts of time and computational resources. While hardware is getting increasingly cheaper and available by the day, the processing of network logs in large batches can still significantly bottleneck the anomaly detection procedure, decreasing its usefulness, especially considering the fact that near real-time reporting of the detected anomalies is imperative for a successful mitigation strategy. In order to fully exploit the added value of machine learning in anomaly detection, batch processing methods will have to be substituted with stream-processing ones that will continuously produce useful analytics to be used in the remediation of the network.

6.2 Future work

As described in the previous section, a module that will match the detected anomalies with specific types of threats or attacks is missing from the capabilities of the engine. This module could be based to a popular supervised learning family of algorithms, namely decision trees. Decision tree learning uses a decision tree as a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves) and is one of the most common predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Such a classification tree could be trained to assign attack labels to network logs, using either existing labelled network datasets with predefined anomalies identified as threats by signature-based IDPSs (e.g. Snort, Symantec), or datasets with artificially added synthetic attacks. Subsequently, the detected anomalies by the Apache Spot engine could be inserted to the tree as input, in order be correlated with types of network attacks.

Moreover, as the popularity of stream data platforms is skyrocketing, the benefits of transitioning from batch processing to a streaming paradigm can be considered a solution to the increasing demands for (near) real-time cybersecurity mitigation and prevention of large-scale attacks. Analytics based on streaming data can enable new types of latency-critical applications and give more actual operational insights. In our case, streaming processing can be achieved via micro-batching, a technique that allows a process or task to treat a stream of data as a sequence of small batches or chunks. For incoming streams, the events can be packaged into small batches and delivered to a batch system for processing.

Of course, a modification of this magnitude will require changes to the machine-learning algorithm as well, in order to update the local model each time new data arrives and also to create the shared model when a pre-defined threshold is met. Despite the challenges, however, online (streaming) learning is crucial for the implementation of real-time analytics, since it minimizes the excessive reaction times, can dynamically adapt to changing data, supports low-latency processing with scaling across multiple nodes and has demonstrated improved accuracy over offline (batch-processing) algorithms.

References

1. Amazon Web Services, (2016), *Big Data Analytics Options on AWS*, online at: https://d0.awsstatic.com/whitepapers/Big_Data_Analytics_Options_on_AWS.pdf
2. Blei D., Ng A., Jordan, M., (2003),. *Latent Dirichlet allocation*, J. Mach. Learn. Res., 3, pp. 993–1022.
3. Cannady J., (1998), *Artificial neural networks for misuse detection*, In: Proceedings of the 1998 National Information Systems Security Conference (NISSC'98), Arlington, VA, pp. 443–456.
4. Chemitiganti V., (2017), *Payment Providers – How Big Data Analytics provide new opportunities in 2017*, Hortonworks, article found online at: <https://hortonworks.com/blog/payment-providers-big-data-analytics-provides-new-opportunities-2017/>
5. Chen M., Mao S., Zhang Y., Leung V., (2014), *Big Data: Related Technologies, Challenges and Future Prospects*, SpringerBriefs in Computer Science, Springer International Publishing.
6. Chen W., Hsu S., Shen H., (2005), *Application of SVM and ANN for intrusion detection*, Computers & Operations Research, pp.2617-2634.
7. Cloud Security Alliance - CSA, (2012), *Security as a Service Working Group*.
8. Columbus L., (2016), *Ten Ways Big Data Is Revolutionizing Marketing And Sales*, Forbes, article found online at: <https://www.forbes.com/sites/louiscolombus/2016/05/09/ten-ways-big-data-is-revolutionizing-marketing-and-sales/#6da7caef21cf>
9. CSO, (2016), *How Big Data is Improving Cyber Security*, article found online at: <https://www.csoonline.com/article/3139923/security/how-big-data-is-improving-cyber-security.html>
10. Dempster A., Laird N., Rubin D., (1977), *Maximum Likelihood from Incomplete Data via the EM Algorithm*, Journal of the Royal Statistical Society, Series B. 39, pp.1–38.
11. Dua S., Du X., (2016), *Data Mining and Machine Learning in Cybersecurity*, CRC Press.
12. Eaton C., Deroos D., Deusch T., Lapid G., Zikopoulos P., (2012), *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, The McGraw-Hill Companies.

13. Eskin E., Arnold A., Prerau M., Portnoy L., Stolfo S., (2002), *A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data*, In: Applications of Data Mining in Computer Security, Chap. 4.
14. Frazzoli E., (2013), *Intro to Hidden Markov Models the Baum-Welch Algorithm*, Aeronautics and Astronautics, Massachusetts Institute of Technology.
15. Gartner Research, (2012), *The importance of "Big Data": A definition*.
16. Gartner, (2013), *Hype Cycle for Emerging Technologies Maps Out Evolving Relationship Between Humans and Machines*, available online at: <https://www.gartner.com/newsroom/id/2575515>
17. Ghosh A., Schwartzbard A., (1999), *A study in using neural networks for anomaly and misuse detection*, In: Proceedings of the 8th USENIX Security Symposium, Washington, DC, pp. 141–152.
18. Gong F., (2003), *Deciphering detection techniques: Part II. Anomaly-based intrusion detection*, McAfee Network Security Technologies Group.
19. Haridas M, (2015), *Redefining Military Intelligence Using Big Data Analytics*, Scholar Warrior article.
20. Howard J, Kazar M., Menees S., Nichols D, Satyanarayanan M., Sidebotham R., West M., (1988), *Scale and performance in a distributed file system*, ACM Transactions on Computer Systems (TOCS), Chapt.6(1), pp.51–81.
21. Hu W., Liao Y., Vemuri V., (2003), *Robust support vector machines for anomaly detection in computer security*, In: Proceedings of the International Conference on Machine Learning, Las Vegas, NV, pp. 282–289.
22. Hutchinson S., (2014), *Semantic Features from Web-Traffic Streams*, Network Science and Cybersecurity, Advances in Information Security, vol 55, Springer.
23. IBM Software, (2013), *Data-driven healthcare organizations use Big Data analytics for big gains*, White Paper.
24. IBM, 2013, *What is Big Data?*, available online at: <http://www-01.ibm.com/software/data/bigdata/>
25. InfoSec Institute, (2013), *The Impact of Cybercrime*.
26. Kaspersky, (2016), *Attack on dyn explained*, available online at: <https://blog.kaspersky.com/attack-on-dyn-explained/13325/>
27. Khan W., (2015), *Big Data is Set to Transform the Consumer Goods Industry*, LinkedIn, article found online at: <https://www.linkedin.com/pulse/big-data-set-transform-consumer-goods-industry-waleed-khan/>

28. Kruegel C., Toth T., (2003), *Using decision trees to improve signature-based intrusion detection*, In: Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection, Pittsburgh, pp. 173–191.
29. Kumar, U., Ahmadi, A., Verma, A. K., Varde, P. (2016), *Current Trends in Reliability, Availability, Maintainability and Safety: An Industry Perspective*, 1st ed., Springer.
30. Lebid M., (2017), *9 Examples of Big Data Analytics in Healthcare That Can Save People*, Datapine, article found online at:
<https://www.datapine.com/blog/big-data-examples-in-healthcare/>
31. Lerner J, Dahl R, Hariri A, Taylor S., (2007), *Facial expressions of emotion reveal neuroendocrine and cardiovascular stress responses*, *Biological Psychiatry*. 61: 253-60.
32. Liao Y., Vemuri V., (2002), *Use of k-nearest neighbor classifier for intrusion detection*, *Computers & Security* 21 (5) pp.439–448.
33. Liu Z., Florez G., Bridges S., (2002), *A comparison of input representations in neural networks: A case study in intrusion detection*, In: Proceedings of the 2002 International Joint Conference on Neural Networks, Honolulu.
34. Lokesak B., (2008), *A Comparison Between Signature Based and Anomaly Based Intrusion Detection Systems*.
35. Mitchell T., (1997), *Machine Learning*, McGraw Hill.
36. Moubayed N., Wall D., McGough S., (2017), *Identifying Changes in the Cybersecurity Threat Landscape Using the LDA-Web Topic Modelling*, Data Search Engine, HAS 2017, vol 10292, Springer.
37. Murty M., Jain A., Flynn P., (1999), *Data clustering: A review*, ACM Computing Surveys, vol. 31.
38. NIST Big Data Workshop, (2013). *NIST Big Data Program*.
39. O'Brien S., (2016), *How Do Telecommunication Companies Use Big Data? 8 Resources to Bookmark*, Datameer, article found online at:
<https://www.datameer.com/company/datameer-blog/8-big-data-telecommunication-use-case-resources/>
40. Oracle, (2012), *Meeting the Challenge of Big Data*, available online at:
<http://www.oracle.com/us/technologies/big-data/big-data-ebook-1866291.pdf>
41. Paganini P., (2013), *Norton Report, the impact of cybercrime according to Symantec*, Security Affairs.
42. Ponemon Institute, (2014), *Global Report on the Cost of Cyber Crime*, HP security, available online at: <http://www8.hp.com/us/en/software-solutions/ponemon-cyber-security-report>

43. Portnoy L., Eskin E., Stolfo S., (2001), *Intrusion detection with unlabeled data using clustering*, In: Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA), Philadelphia
44. Qiao, Y., Xin X., Bin Y., Ge S., (2002), *Anomaly intrusion detection method based on HMM*, Electronics Letters 38, pp.663–664.
45. Rabiner L., (1989), *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE 77, pp.257–286.
46. Samtani S., Chinn R., Chen H., (2015), *Exploring Hacker Assets in Underground Forums*, Intelligence and Security Informatics, pp.13-36.
47. SANS Institute, (2016), *Using Analytics to Predict Future Attacks and Breaches*, White Paper, SAS.
48. Shang W., Jiang Z. M., Hemmati H., Adams B., Hassan A.E., Martin P., (2013), *Assisting developers of Big Data analytics applications when deploying on Hadoop clouds*, in Proceeding of the international conference on software engineering, vol. 203, pp. 402-411.
49. Shukla D., (2017), *The deployment of Big Data and analytics techniques to reinforce security*, CustomerThink, article found online at: <http://customerthink.com/the-deployment-of-big-data-and-analytics-techniques-to-reinforce-security/>
50. Sivarajah U., Kamal MM. et al., (2017), *Critical Big Data Analysis Challenges and Analytical Methods*, Journal of Business Research, 70, 263-286.
51. Sparapani T., (2017), *How Big Data And Tech Will Improve Agriculture, From Farm To Table*, Forbes, article found online at: <https://www.forbes.com/sites/timsparapani/2017/03/23/how-big-data-and-tech-will-improve-agriculture-from-farm-to-table/#4666b5d15989>
52. Sravanthi K., Reddy T., (2015), *Applications of Big Data in Various Fields*, International Journal of Computer Science and Information Technologies, Vol. 6 (5), pp. 4629-4632.
53. Srivastava U., Gopalkrishnan S., (2015), *Impact of Big Data Analytics on Banking Sector: Learning for Indian Banks*, Procedia Computer Science 50, 643 – 652
54. Talabis M., McPherson R., Miyamoto I., Martin J., (2014), *Information Security Analytics: Finding Security Insights, Patterns, and Anomalies in Big Data*, Syngress Media.
55. UN, (2012), *Big Data for Development: Challenges & Opportunities*, available online at: <http://www.unglobalpulse.org/sites/default/files/BigDataforDevelopment-UNGlobaIPulseJune2012.pdf>

56. Wang W., Guan X., Zhang X., Yang L., (2006), *Profiling program behaviour for anomaly intrusion detection based on the transition and frequency property of computer audit data*, Computers & Security 25, pp.539–550.
57. Warrender C., Forrrest S., Pearlmutter B., (1999), *Detecting intrusions using system calls: Alternative data models*, IEEE Symposium on Security and Privacy, Oakland, pp. 133–145.
58. Waterman K, Bruening P., (2014), *Big Data Analytics: Risks and Responsibilities*, International Data Privacy Law 4.
59. WEF, (2012). *Big Data, big impact: New possibilities for international development*, White Paper at Davos, available online at: <http://www.weforum.org/reports/big-data-big-impact-new-possibilities-international-development>
60. White House, (2012), *Big Data Research and Development Initiative*, available online at: https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf
61. Wolfson R., (2015), *How an enterprise begins its big data journey*, O'Reilly, article found online at: <https://www.oreilly.com/ideas/how-an-enterprise-begins-its-big-data-journey>
62. Wolpert D., Macready W., (1995), *No Free Lunch Theorems for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute

Utilised technologies

1. Apache Hadoop: <https://hadoop.apache.org/>
2. Apache Hive: <https://hive.apache.org/>
3. Apache Kafka: <https://kafka.apache.org/>
4. Apache Spark: <https://spark.apache.org/>
5. Apache Spot: <http://spot.incubator.apache.org/>
6. BoNeSi DDoS Botnet Simulator: <https://github.com/Markus-Go/bonesi>
7. BreakingPoint Network security testing platform: <https://www.ixiacom.com/products/network-security-testing-breakingpoint>
8. Cloudera Distribution for Hadoop: <https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html/>
9. Facebook Threat Exchange: <https://developers.facebook.com/products/threat-exchange>
10. Flux: <https://facebook.github.io/flux/>

11. GraphQL: <http://graphql.org/learn/>
12. Ipython: <https://ipython.org/>
13. McAfee Global Threat Intelligence (GTI):
<https://kc.mcafee.com/corporate/index?page=content&id=KB70130/>
14. React JS: <https://facebook.github.io/react/>

All websites were last accessed on July 2017.

List of Acronyms

Acronym	Meaning
API	Application Programming Interface
CDH	Cloudera Distribution for Hadoop
DARE	Data Analysis and Remediation Engine
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IDPS	Intrusion Detection and Prevention System
IoT	Internet of Things
IPS	Intrusion Prevention System
ISP	Internet Service Provider
LDA	Latent Dirichlet Allocation
LXC	LinuX Container
NFV	Network Function Virtualisation
PoP	Point of Presence
SIEM	Security Information and Event Management
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
VM	Virtual Machine

vNSF	virtual Network Security Function
vNSFO	vNSF Orchestrator
VPN	Virtual Private Network

Appendix

Apache Spot Installation Procedure

Here, a step-by-step documentation of Apache Spot installation is included. Since Spot is still being developed and relies on a lot of third party software packages, there are a lot of issues still being resolved as commits to Spot's code are frequent. Spot documentation and instructions can be found on:

<http://spot.incubator.apache.org/doc/>

Spot code and further explanation of setup instructions and configuration files can be found on:

<https://github.com/apache/incubator-spot>

Refreshing the package handler

It is highly recommended that you refresh the package handler across all three nodes. For our Ubuntu installation we used:

```
sudo apt-get update
sudo apt-get upgrade
```

Configuring users and permissions in the cluster

```
sudo addgroup --gid 2000 supergroup
```

Create a username and password (we used "spotuser") in all three nodes:

```
sudo useradd -uid 2001 -s /bin/bash -m spotuser
passwd spotuser
su spotuser
```

```
nano .profile
```

The following lines were added in `.profile` for convenience, in order to shorten commonly used commands:

```
unalias fs &> /dev/null  
alias fs="hdfs dfs"  
unalias hls &> /dev/null  
alias hls="hdfs dfs -ls"
```

For the changes to take effect we run the following in bash:

```
source .profile
```

At this point we recommend to include `spotuser` in the `sudoers` list to make the installation a bit easier.

Setup on Edge Gateway

First we install `git` so we can clone `Spot`:

```
apt-get install git  
git clone https://github.com/apache/incubator-spot.git  
cd incubator-spot/spot-setup/
```

`Spot` includes the `spot.conf` configuration file which needs to be edited to suit the specific installation as seen in Figure A.1:

```
nano spot.conf
```

```

GNU nano 2.2.6      File: /etc/spot.conf
node configuration
INODE='10.101.10.21'
LNODE='10.101.10.21'
MNODE='10.101.10.25'
BNAME='ubuntu'
} Configuration of the node IPs

hdfs - base user and data source config
HUSER='/user/ubuntu'
HNS_PATH=${HUSER}/${DSOURCE}/hive/y=${YR}/m=${MH}/d=${DY}/
ROXY_PATH=${HUSER}/${DSOURCE}/hive/y=${YR}/m=${MH}/d=${DY}/
LOW_PATH=${HUSER}/${DSOURCE}/hive/y=${YR}/m=${MH}/d=${DY}/
FLOW_PATH=/user/ubuntu/flow/flow/binary/20170614/12/
PATH=${HUSER}/${DSOURCE}/scored_results/${FDATE}

impala config
IMPALA_DEM='10.101.10.23'

kerberos config
RB_AUTH=false
INITPATH=
INITOPTS=
KEYTABPATH=
RB_USER=

local fs base user and data source config
HUSER='/home/ubuntu'
HNS_PATH=${HUSER}/ml/${DSOURCE}/${FDATE}
HNS_PATH=${HUSER}/ipython/user/${FDATE}
HNS_PATH=${HUSER}/ingest

dns suspicious connects config
SER_DOMAIN=''

SPK_EXEC='4'
SPK_EXEC_MEM='512m'
SPK_DRIVER_MEM='512m'
SPK_DRIVER_MAX_RESULTS='800m'
SPK_EXEC_CORES='1'
SPK_DRIVER_MEM_OVERHEAD='50'
SPK_EXEC_MEM_OVERHEAD='50'
OL='1e-6'
} Configuration of the Spark executor number, memory resources etc.

OPIC_COUNT=20
UPFACTOR=1000
[ Read 44 lines (Warning: No write permission) ]
Get Help  WriteOut  Read File  Prev Page  Cut Text  Cur Pos
Exit      Justify   Where Is   Next Page  Uncut Text To Spell

```

Figure A.1: Example of spot.conf, highlighting the main configurations that need to be altered

We set up Spot with 4 Spark Executors utilizing up to 8GBs of memory. The exact number depends on the characteristics of the infrastructure hosting the VMs.

```

SPK_EXEC=''          ---> Maximum number of executors
SPK_EXEC_MEM=''     ---> Memory per executor in MB i.e. 30475m
SPK_DRIVER_MEM=''   ---> Driver memory in MB i.e. 39485m
SPK_DRIVER_MAX_RESULTS='' ---> Maximum driver results in MB or GB i.e. 8g
SPK_EXEC_CORES=''   ---> Cores per executor i.e. 4

```

```
SPK_DRIVER_MEM_OVERHEAD=' ' ---> Driver memory overhead in MB i.e. 3047. Note
that there is no "m" at the end.

SPK_EXEC_MEM_OVERHEAD=' ' ---> Executor memory overhead in MB i.e. 3047. Note
that there is no "m" at the end.

SPK_AUTO_BRDCST_JOIN_THR='10485760'--->Spark's
spark.sql.autoBroadcastJoinThreshold. Default is 10MB, increase this value to
make Spark broadcast tables larger than 10 MB and speed up joins.

PRECISION='64' ---> Indicates whether spot-ml is to use 64
bit floating point numbers or 32 bit floating point numbers when representing
certain probability distributions.
```

Spot git documentation explains how to choose values for optimal performance in your infrastructure for Spark. In our example we use testing values for our cluster, not the optimal. Next, we need to copy `spot.conf` to all nodes, in the `/etc` folder.

```
cp -a spot.conf /etc
su - hdfs
hdfs dfs -mkdir /user/spotuser
hdfs dfs -chown spotuser:supergroup /user/spotuser
```

On the setup-ingest node (host-2) run `./hdfs_setup.sh` shell script and comment out the lines referring to Hadoop (lines 27-28 in the following figure):

```
10 #
11 # http://www.apache.org/licenses/LICENSE-2.0
12 #
13 # Unless required by applicable law or agreed to in writing, software
14 # distributed under the license is distributed on an "AS IS" BASIS,
15 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
16 # See the license for the specific language governing permissions and
17 # limitations under the license.
18 #
19
20 DSOURCES=('flow' 'dns' 'proxy')
21 DFOLDERS=('binary' 'hive' 'stage')
22 source /etc/spot.conf
23
24 #
25 # creating HDFS user's folder
26 #
27 hadoop fs -mkdir ${HUSER}
28 hadoop fs -chown ${USER}:supergroup ${HUSER}
29
30 for d in "${DSOURCES[@]}"
31 do
32     echo "creating /$d"
33     hadoop fs -mkdir ${HUSER}/$d
34     for f in "${DFOLDERS[@]}"
35     do
36         echo "creating $d/$f"
37         hadoop fs -mkdir ${HUSER}/$d/$f
38     done
39 done
40
41 #
42 # create hive tables
43 #
44 #configure / create catalog
45 hive -e "CREATE DATABASE ${DBNAME}"
46
47 for d in "${DSOURCES[@]}"
48 do
49     hive -hiveconf huser=${HUSER} -hiveconf dbname=${DBNAME} -f create_${d}_avro_parquet.hql
50 done
```

Figure A.2: Example of hdfs_setup.sh script, showing which lines to comment out.

Ingest on Edge Gateway

In the spot-ingest folder, we create a src folder that will hold the sources solving various Spot dependencies. Depending on the spot version, the src folder might already exist. If it exists, mkdir is not required.

```
cd incubator-spot/spot-ingest/  
mkdir src  
cd src
```

First we install the Python Package Installer (pip) and then use pip to install various dependencies.

```
wget --no-check-certificate https://bootstrap.pypa.io/get-pip.py  
sudo -H python get-pip.py  
sudo -H pip install kafka-python  
sudo -H pip install watchdog  
sudo apt-get install build-essential  
sudo apt-get install wireshark  
git clone https://github.com/Open-Network-Insight/spot-nfdump.git  
cd spot-nfdump  
./install_nfdump.sh  
cd ..  
cd common  
wget https://repol.maven.org/maven2/org/apache/spark/spark-streaming-kafka-0-8-assembly\_2.11/2.0.0/spark-streaming-kafka-0-8-assembly\_2.11-2.0.0.jar  
cd ..  
nano ingest_conf.json
```

Next we edit ingest_conf.json to set up the directory for storing netflow, dns and proxy data and the related staging areas required by the ingestion workers. In

Create the following folders in `/home/spotuser`:

```
mkdir traffic
mkdir traffic/flow
mkdir traffic/dns
mkdir traffic/proxy
mkdir traffic/flow/local_staging
mkdir traffic/dns/local_staging
mkdir traffic/dns/split_staging
```

These will be used in the configuration of ingest_conf.json

```
spotuser@cloudera-host-2: ~/incubator-spot/spot-ingest
GNU nano 2.2.6 File: ingest_conf.json

"dbname" : "spot",
"hdfs_app_path" : "/user/spotuser",
"collector_processes":5,
"ingestion_interval":1,
"spark-streaming":{
  "driver_memory":"8000m",
  "spark_exec":"1",
  "spark_executor_memory":"8000m",
  "spark_executor_cores":"2",
  "spark_batch_size":"1000"
},
"kafka":{
  "kafka_server":"10.101.30.12",
  "kafka_port":"9092",
  "zookeeper_server":"10.101.30.10",
  "zookeeper_port":"2181",
  "message_size":900000
},
"pipelines":{
  "flow":{
    "type":"flow",
    "collector_path":"/home/spotuser/traffic/flow",
    "local_staging":"/home/spotuser/traffic/flow/local_staging",
    "supported_files":["nfcap."],
    "process_opt":""
  },
  "dns":{
    "type":"dns",
    "collector_path":"/home/spotuser/traffic/dns",
    "local_staging":"/home/spotuser/traffic/dns/local_staging",
    "supported_files":[".pcap"],
    "pkt_num":"650000",
    "pcap_split_staging":"/home/spotuser/traffic/dns/split_staging",
    "process_opt":"-E separator=-, -E header=y -E occurrence=f -T fields -e frame.time_epoch -e frame.len -e ip.src -e ip.dst -e dns.resp.name -e dns.resp.type -e d"
  },
  "proxy":{
    "type":"proxy",
    "collector_path":"/home/spotuser/traffic/proxy",
    "supported_files":[".log"],
    "parser":"bluecoat.py"
  }
}
}

Get Help WriteOut Read File Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page UnCut Text To Spell
```

Figure A.3: The ingest_conf.json file, highlighting the changes we made.

In the folder pipelines/flow/ replace worker.py with our modified version found in:

<https://drive.google.com/open?id=0B239aXxSir9QOXN3N1pqUGxQNTg/>

In the pipelines/dns folder replace worker.py with our modified version:

<https://drive.google.com/file/d/0B239aXxSir9Qd0g2OVpBV1BOQm8/view>

Anomaly detection on Machine-Learning (ML) Node

At this point we should mention that due to the fact that Apache Spot is under development we decided to use a version that we were familiar with and have uploaded the files that we used in Google Drive. An important issue is that the latest Spot version uses Spark 2.1 for the ML part while the latest Cloudera version is compatible with Spark 1.6. Download the ml folder from this link and place it in the ML node:

<https://drive.google.com/drive/folders/0B239aXxSir9QMG9iTFB5S3BKSkk>

To install dependencies, connect to the ML node, and use the following:

```
cd ml/src
sudo apt-get install python-software-properties
add-apt-repository ppa:webupd8team/java
sudo apt-get install oracle-java8-installer
type -p javac | xargs readlink -f | xargs dirname | xargs dirname
cd ..
sbt assembly
```

Visualisation on Operational Analytics (OA) Node

In the operational analytics node:

```
cd spot-oa
sudo apt-get install python-pip python-dev
sudo pip install -r requirements.txt
cd context
cd ..
nano oa/components/data/engine.json
```

```
spotuser@cloudera-manager: ~/spot-oa/oa/components/data
GNU nano 2.2.6 File: engine.json
"oa_data_engine":"impala",
  "impala":{
    "impala_daemon":"10.101.30.10"
  },
  "hive":{}
}
```

Figure A.4: Engine.json configuration, depending on the use of impala or hive as the database engine.

Install dependencies:

```
cd ui
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install -y nodejs
sudo npm install browserify -g
sudo npm install uglifyjs -g
npm install
cd ..
./runIpython.sh
```