Scalable and Efficient Network Anomaly Detection on Connection Data Streams

Aniss Chohra

A Thesis

in

The Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

May 2019

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Aniss Chohra**

Entitled: **Scalable and Efficient Network Anomaly Detection on Connection Data Streams**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

Chair
Dr. Jun Yan

External Examiner
Dr. Olga Ormandjieva

Examiner
Dr. Chun Wang

Supervisor
Dr. Mourad Debbabi

Approved by
Abdessamad Ben Hamza, Chair
of Concordia Institute for Information Systems Engineering (CIISE)

Wednesday, April 10th, 2019

Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Scalable and Efficient Network Anomaly Detection on Connection Data Streams

Aniss Chohra

Everyday, security experts and analysts must deal with and face the huge increase of cyber security threats that are propagating very fast on the Internet and threatening the security of hundreds of millions of users worldwide. The detection of such threats and attacks is of paramount importance to these experts in order to prevent these threats and mitigate their effects in the future. Thus, the need for security solutions that can prevent, detect, and mitigate such threats is imminent and must be addressed with scalable and efficient solutions. To this end, we propose a scalable framework, called *Daedalus*, to analyze streams of NIDS (network-based intrusion detection system) logs in near real-time and to extract useful threat security intelligence. The proposed system pre-processes massive amounts of connections stream logs received from different participating organizations and applies an elaborated anomaly detection technique in order to distinguish between normal and abnormal or anomalous network behaviors. As such, *Daedalus* detects network traffic anomalies by extracting a set of significant pre-defined features from the connection logs and then applying a time series-based technique in order to detect abnormal behavior in near real-time. Moreover, we correlate IP blocks extracted from the logs with some external security signature-based feeds that detect factual malicious activities (e.g., malware families and hashes, ransomware distribution, and command and control centers) in order to validate the proposed approach. Performed experiments demonstrate that *Daedalus* accurately identifies the malicious activities with an average F_1 score of 92.88%. We further compare our proposed approach with existing K-Means and deep learning (LSTMs) approaches and demonstrate the accuracy and efficiency of our system.

Acknowledgments

My deep gratitude goes first to my supervisor, Dr. Mourad Debbabi, who guided me through my whole graduate program and was of helpful and amazing support.

I would also like to extend my gratitude to all my laboratory colleagues who shared with me the wonderful experience of this research project and were of great support and advices.

In addition to that, I would like to express my sincere and grateful gratitude for all my family members for their continuous support and encouragement during all this time.

Last but not least, I would like to express my thanks to the Benmoussa family members who were very helpful and supportive since I arrived to Canada and during my graduate program.

List of Abbreviations

ANN Artificial Neural Network

BM Boltzmann Machine

CNN Convolutional Neural Network

CPU Central Processing Unit

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

EWMA Exponential Weighted Moving Average

HIDS Host-based Intrusion Detection System

HTTP Hyper-Text Transfer Protocol

HTTPS Secure Hyper-Text Transfer Protocol

IDS Intrusion Detection System

IoC Indicator of Compromise

IP Internet Protocol

KNN K-Nearest Neighbor

LSTM Long-Short Term Memory

NIDS Network-based Intrusion Detection System

PCA Principal Component Analysis

PSO Particle Swarm Optimization

RNN Recurrent Neural Network

SANN Supervised Artificial Neural Network

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SVM Support Vector Machine

SSL Secure Socket Layer

TCP Transmission Control Protocol

UANN Unsupervised Artificial Neural Network

UDP User Datagram Protocol

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, we present the motivations of our work. Then, we formally define the problem that we try to solve, state our objectives and research contributions. Finally, we give an overview of this thesis organization.

1.1 Motivations

During the last decade, a huge increase in the number of cyber threats and security attacks has been observed ranging from ransomware attacks to denial of service attacks and botnets, and from social engineering threats to data breach threats, which poses a real threat to millions of users, making the prevention, mitigation, and detection of such attacks a challenging and difficult task for security analysts and experts. As an example to such threats, The *Wannacry Ransomware* [74] released by a hacking group named *Shadow Brokers*, propagated worldwide during May 2017, and had devastating consequences in several countries affecting hundreds of thousands of machines and many organizations, such as Cambrian College in Canada and Saudi Telecom company. Another example is The *Mirai Botnet* [15] which also had a severe impact worldwide in late 2016 mostly in the U.S by affecting vulnerable Internet of Things devices and turning them into a zombie army.

In order to detect these threats in fast and quick time delays, security analysts have a wide range of tools and frameworks such as: social networks where users react to worldwide attacks and vulnerabilities in real-time, intrusion detection and prevention tools that monitor (log) network traffic activities in human readable formats, and many others. Although intrusion detection systems offer network activity monitoring in real-time, they suffer from some limitations such as the tremendous lag to learn novel attcks signatures and add them to the dedicated databases. In addition to that, these databases tend to grow exponentially in size thus leading to decrease their access time. Therefore, the need of security experts for techniques and approaches to take advantage of these logged data, and generate useful cyber threat intelligence has arisen. These techniques must not only be able to detect known attack patterns or signatures, but also detect zero-day attacks, all of this in a scalable and less resources consumption fashion. As an example of proposed techniques, anomaly detection is a field of interest that has attracted many researchers due to its ability to solve some of the issues mentioned above.

1.2 Problem Statement

Nowadays, many attacks and threats are launched around the globe to such an extent that detecting and monitoring all of them is not an easy and straightforward task. Moreover, the time interval between the occurence of these threats and their detection by cyber experts and analysts can be very critical to the security of cyber infrastructures. Security analysts need to be aware of what is happening on their organization's network in near real-time fashion. Thus, they need tools that enable them to monitor all the network traffic and activity that is occuring in a quickly time delays. Intrusion detection systems (IDS) are the most used tools to achieve the aforementioned goal by logging all targeted network traffic by security experts in a human readable and comprehensive manner. However, these IDS tools are not enough to detect all the attacks within the network since they do not come with the fully detection functionalities especially for unknown attacks signatures. Therefore, security analysts and experts needed to develop and come-up with new techniques in order for them to leverage these logged activities and increase the detection rate of such attacks. Anomaly detection is one of the most explored and studied fields in order to achieve the aforementioned objective. Many approaches and algorithms have been proposed for anomaly detection task falling into many paradigms (machine learning, statistical, soft-computing, etc), and each one of them has its advantages over other techniques and its limitations and flaws. But, the majority of these anomaly detection techniques suffer from scalability issues due to the fact that they deal in most cases with small datasets (Benchmark datasets), and it is very rare to find one of these techniques experimenting with huge network traffic. To address these challenges, we designed and developed a framework called *Daedalus* capable of generating cyber threat intelligence from a stream of connections by detecting anomalies in a scalable way, geolocate threat actors, and correlate this intelligence with external and factual security sources. *Daedalus* uses a combination of several algorithms and is splitted into four major steps: *feature extraction* where we select the appropriate set of features needed and apply the required preprocessing on them to get the observed network behavior, *adaptive thresholding* where we define a dynamic and adaptive threshold for each of these extracted features, *predictive analysis* where we propose our own predictive algorithms, which tries to remove noisy patterns from the observed network behavior, and *anomaly detection* step where we compute anomalies scores and make a decision based on them in order to decide whether a feature is anomalous or not. Our tool aims to fix this scalability issue by leveraging the advantages of some state of the art techniques and improve the execution times, CPU, and RAM consumption.

1.3 Objectives and Contributions

The main objective of this research is to build a scalable tool and framework capable of analyzing massive stream of connection logs from different sources, aggregate them in a smart and useful way, and detect malicious and abnormal network activities. To this end, we aim to achieve the following complimentary objectives:

a) Analyze massive connection streams to generate in an efficient and scalable way threat intelligence that can be used in the detection, identification, mitigation, and attribution of cyber threats.

b) Correlate this generated intelligence with other security sources of indicators of compromise (IoCs) in order to have more details about the detected malicious activity, such as malware family or malware hashes.

c) Validate the proposed technique through extensive experimentation on real-life IDS streams.

By fully achieving these objectives, we would be able to help security analysts to analyze streams of connections logs in more details and take security decisions based on the generated threat intelligence; for instance, blacklisting some malicious IP blocks.

Our contributions are summarized as follows:

a) Comparative study of the state of the art proposals in network anomaly detection on connection stream data.

b) Design and elaboration of an innovative technique for network anomaly detection on connection stream data that achieves high efficiency and scalability.

c) Design and implementation of a framework that analyzes massive real-life connection stream data.

1.4 Thesis Organization

The remainder of this thesis is organized as follows: chapter 2 provides the necessary background and the litterature review of the most prominent related work. Chapter 3 presents the time series based technique used for anomaly detection. Chapter 4 gives details about the proposed system including the implementation details (code samples), the technology stack describing all the technologies used, and the built web portals containing the user-friendly dashboards. In addition to that, we also detail the steps to evaluate our approach. Chapter 5 concludes this thesis with some remarks and notes and discusses possible future works.

Chapter 2

Background and State of the Art

In this chapter, we first introduce the needed background for our work including an overview of intrusion detection systems (IDSs), and an overview of BRO NIDS tool. Then, we state the most relevant works that have been proposed and published in the field of network anomaly detection.

2.1 Background

In this section, first we give a general description of intrusion detection systems (IDS) and compare between two main types of IDS; namely: *host-based intrusion detection systems (HIDS)* and *network-based intrusion intrusion detection systems (NIDS)*. Next, we are going to have a detailed view of Bro NIDS in which we describe its architecture and design. Then, we present a detailed state of the art of existing network anomaly detection techniques.

2.1.1 Intrusion Detection Systems

In parallel with the continious growth of the Internet and Information Systems, novel attempts to attack these networks and tools and exploit possible their vulnerabilities also have increased tremendously. Thus, making the detection, mitigation, and prevention of such threats very important for every security expert. Intrusion detection systems (IDS) are the most used tools in order to achieve the above goals. Intrusion detection system (IDS) can be any software or device which monitors networks and devices activities in order to detect suspicious activities. Thus, intrusion detection systems' main goal is to prepare the background for any security expert to deal with possible threats by:

a) Collecting and monitoring information from different systems and networks,

b) Make them suitable for further analysis in order to detect possible security policies breaches.

Also, IDS systems provide the following functionalities:

a) Monitoring and analysing different users' activities.

b) Security assessment of different vulnerabilities found on the network.

c) Statistical overview of network/users activities patterns and matching them to known attack patterns.

d) Detection of anomalous activities by analysing the flowing traffic over the monitored network.

But in most cases these systems are not enough to completely protect from all types of attacks; especially when all Internet services are publicly available with in most cases their open-source codes/implementations making it easier for attackers to:

a) evade detection techniques by mimicing in some cases normal activities behaviors like in *Mimicry Attacks*,

b) or in other cases implementing novel attacks which do not have known signatures detected by these IDS systems (*zero-day attacks*).

Thus, the evaluation of these IDSs' efficiency and performance systems is of paramount importance for security experts before completely deploying them on the targeted network infrastructure. In this context, three metrics/criterias are used to evaluate them:

a) **Accuracy** which represents the IDS system ability to detect and flag attacks without the presence of false positives (flagging of benign activities as threats).

b) **Scalability** is the most important of these three metrics as it represents the IDS system's speed and rate at auditing and monitoring network/users activities; the more the scalability metric is better, the closest the detection functionality is to *real-time*.

c) **Resistance to Threats and Vulnerabilities** which represents the ability of the IDS system itself to resist to possible attacks and evasion techniques.

2.1.1.1 Host-based versus Network-based Intrusion Detection Systems

Intrusion detection systems (IDSs) can be categorized into two main categories depending on the site of their deployement in addition to other diffrences.

a) ***Host-based Intrusion Detection Systems*** are the first category of IDS that have been proposed since the first designs of IDSs. In HIDS environments, the systems to protect are mainly computer stations; where on each computer station an HIDS system/client is deployed that monitors the computer's system state for any suspicious and critical changes and these clients communicate the logs that they collected with a central HIDS server which is generally located on the internal side of the local network to monitor (behind the firewall); like depicted in Figure 2.1.

The main advantage of HIDS is that they allow the local detection of possible attacks/threats or intruders before their propagation to the whole network infrastructure. In addition to that, and in regard to the deployement of distributed attacks, the need of implementing a communication prototype that allows different workstations to exchange their HIDS information in order to detect such distributed attacks. However, HIDS systems have a major flaw which consists of their limitation in detecting attacks on the infrastructure network itself; especially with the widespreading of wide ranged attacks like *distributed denial of service attacks (DDoS).*

b) ***Network-based Intrusion Detection Systems*** on the other hand are deployed in order to detect attacks on the whole targeted infrastructure's network. Thus, they are generally deployed on the side of targeted network's firewall/router in order to log all incoming and outgoing network traffic; like shown in 2.2.

These are generally deployed on the network's firewall level and analyse the incoming and outgoing network traffic in order to detect known attacks behaviors/patterns. There are many differences between these two categories of IDS systems; which are shown in 2.1:

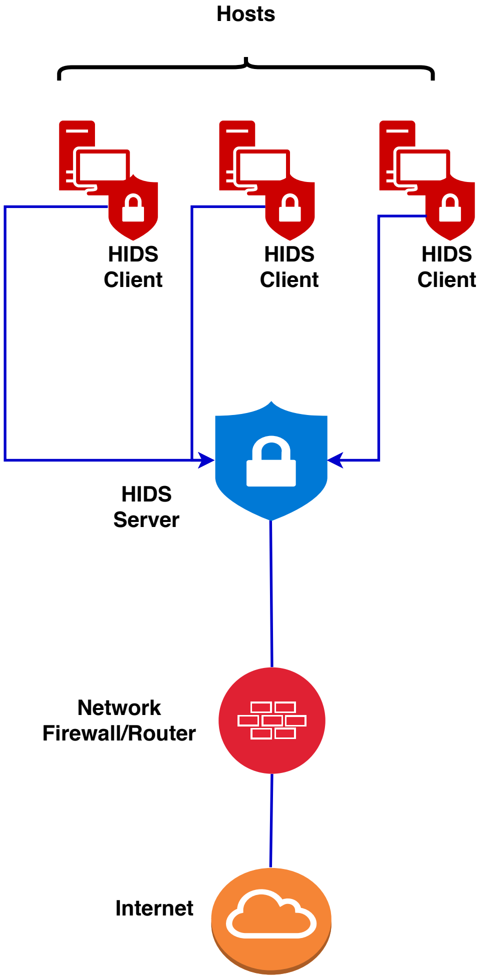


Figure 2.1: Architecture of HIDS.

Table 2.1: Comparative Table Between HIDS and NIDS

Functionality	**HIDS**	**NIDS**	**Description**
Cost and Price	✓	-	HIDS are more affordable and easy to deploy.
Required Training	✓	-	HIDS are straightforward to understand whilst NIDS require deep knowledge.
Deployement Site	Locally	Network's Firewall	HIDS are deployed on the computer side whilst NIDS are deployed on the network's tap.
Bandwidth Requirements	Low	High	NIDS requires more bandwidth since it captures the whole network's traffic.
Cross-platform Compatibility	-	✓	NIDS are more adaptable to other platforms.
Packet Rejection	-	✓	Only NIDS have the capability to reject some packets.
Central Management	✓	✓	Both of them require central management.
Accuracy and False Alarms Rates	✓	High False Alarms Rates	NIDS capture all network's traffic without prior filtering; increasing the number of false alarms.

2.1.2 Bro Network-based Intrusion Detection System

Bro is a powerful open source UNIX-based Network Intrusion Detection System (NIDS) that is most used to detect network behavioral anomalies for cybersecurity purposes. Bro is designed in a way that can resist external threats and attacks against itself and is an event-based Intrusion Detection System that works by

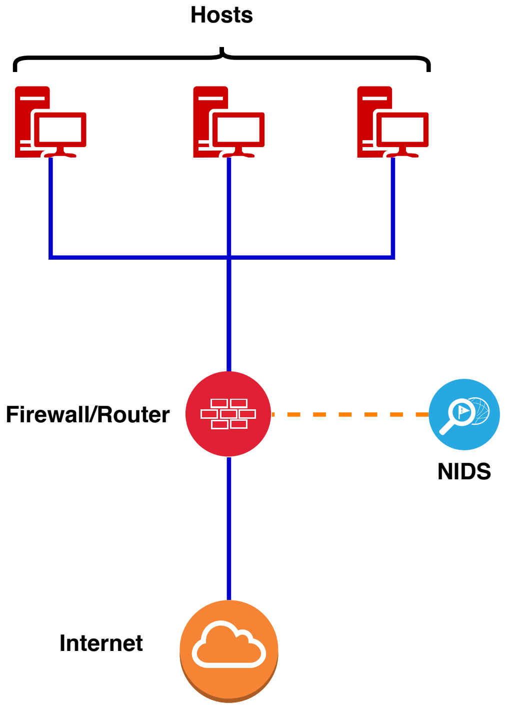


Figure 2.2: Architecture of NIDS.

separating its policy from its mechanisms. Bro comes as a package with several analyzers like protocol analyzer for the majority of most known network TCP/UDP protocols. By analyzing network traffic, Bro generates an abstraction of each activity analyzed as an event. In addition to that, Bro offers its own scripting language that allows its own users to define event-based policies depending on their security requirements [76].

2.1.2.1 Evolution and History

Bro NIDS was originally proposed and developed by Vern Paxson from *ICSI's Center for Internet Research (ICIR), Berkley*. The project was initiated in 1995 and is still being improved till now. *Lawrence Berkley National Laboratory (LBNL)* started officially using the IDS in 1996, and *version 0.2* applying the first set of changes and required updates to the IDS was launched in 1997. Moreover, in 1998, Vern Paxson published a paper in USENIX [66] in which he gave a detailed description of Bro's architecture and design. On the same year, *version 0.4* was released which added the functionality of handling and analyzing HTTP protocol, scan detection, IP fragmentation, and Linux platform support. Just after that, and more precisely in 1999, *version 0.6* was released allowing the open source IDS to handle regular expressions and analyze users' login sessions. Futhermore, in 2000, Vern Paxson published another paper [88] in which an algorithm that is efficiently able to detect *stepping stones* which is an evasion technique used by attackers to keep their anonymity while achieving their goals. During the same year, *version 0.7a90* was released which added important changes to the tool. The next year (2001), another version was released (*v0.7a90*) which added profiling and states management capabilities. In 2002, *version 0.7a175/0.8aX* was released in which attacks signatures, *Simple Mail Transfer Protocol (SMTP)* analyzer, IP version 6, and user manual were added to the framework. The coming year (2003), *version0.8a37* was released which came with communication utilities, database persistence capability, and logs rotation. During the same year, anonymity capability, active mapping, and contextual signatures were also added. In 2004, *versions 0.8aX and 0.9aX* were released enabling the IDS to handle and analyse more protocols like *Secure Socket Layer (SSL)* and *Server Message Block (SMB)*, and also *BroLite* which contains the base configuration of Bro. Moreover, in 2005, *version 1.0* was released introducing *BinPAC* which is a high lever language that is used to describe network protocols parsers and generates C++ code. In addiction to that, and within the same year, *Internet Relay Chat (IRC)* and *Remote Procedure Call (RPC)* analyzers were added to the tool alongside with 64-bit architecture compatibility. Another DIMVA paper [26] was published during the same year which proposed an approach that leverages the advantages of both *host-based* and *network-based* IDSs; by preserving the advantages of network-based ones and improving its weaknesses using specific host-based traffic analysis context. During the next year (2006), *versions 1.1 and 1.2* were released adding *when statement* to the scripting language, resource tuning, *Broccoli* which is: *BRO Client Communication Library* and allowed Bro's users to code applications and tools that are able to communicate in the same language used by the communication protocol of Bro IDS, and *Dynamic Protocol Detection (DPD)* which instead of deciding which protocol analyzer to use (based on the connection's destination port), attributes an *analyzer tree* for each logged connection. In 2007, *version 1.3* was released in which *Ctor expressions*, *GeoIP*, and connections compressor were integrated to the tool. Additionally, and within the same year, *Bro Clustering* was also introduced. In 2008, *version 1.4* added changes and updates regarding *Dynamic Host Configuration Protocol (DHCP)*, *BitTorrent*, *HTTP Entities*, introduced *NetFlow* to Bro, and *Autotuning* capability. The following year (2009), *version 1.5* introducing *BroControl* which consists of a tool that makes Bro IDS straightforward. In 2010, the *National Science Foundation (NSF)* and more precisely the *Office of Advanced Cyberinfrastructure (OAC)* awarded the improvement that had been done on Bro IDS. The next year (2011), *version 2.0* introduced new scripts to the scripting language of Bro, and just after that (in 2012), another version was released (*version 2.1*) which introduced some updates and changes regarding IPv6 support and the *Input Framework* that allowed Bro users to import external data into Bro tool. Last but not least, during the year of 2013 *File analysis* was introduced within *version 2.2* and connections *Summary Statistics*. Also, during the same year, the *National Science Foundation (NSF)* and more precisely the *Office of Advanced Cyberinfrastructure (OAC)* established a *Bro Center of Expertise* which is set to be a central point of communication between diffrent Bro communities worldwide in order to leverage Bro technology and expertise at the best. Figure 2.3 depicts this timeline evolution [76].

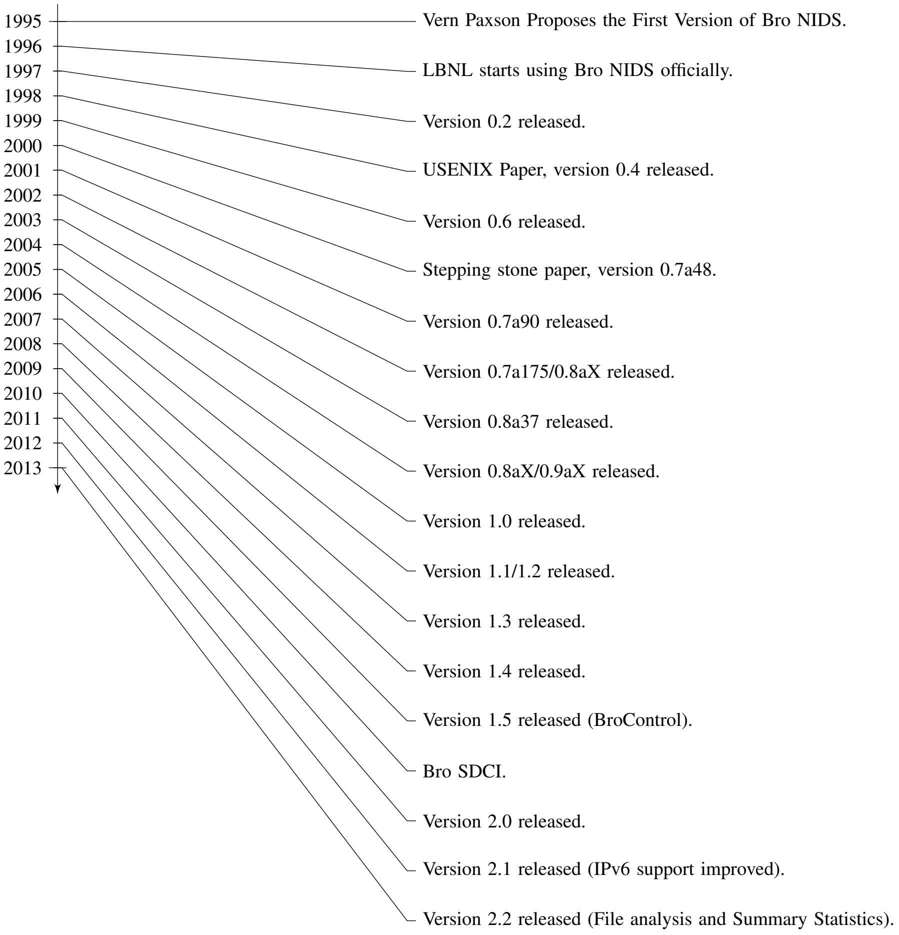

1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
Vern Paxson Proposes the First Version of Bro NIDS.
LBNL starts using Bro NIDS officially.
Version 0.2 released.
USENIX Paper, version 0.4 released.
Version 0.6 released.
Stepping stone paper, version 0.7a48.
Version 0.7a90 released.
Version 0.7a175/0.8aX released.
Version 0.8a37 released.
Version 0.8aX/0.9aX released.
Version 1.0 released.
Version 1.1/1.2 released.
Version 1.3 released.
Version 1.4 released.
Version 1.5 released (BroControl).
Bro SDCI.
Version 2.0 released.
Version 2.1 released (IPv6 support improved).
Version 2.2 released (File analysis and Summary Statistics).

Figure 2.3: Bro Evolution and History Timeline [76, 3].

2.1.2.2 Architecture and Implementation

As shown in Figure 2.4, Bro NIDS has been designed as a set of three layers: ***packet capture***, ***policy-neutral event engine***, and ***policy layer*** [76, 3]. These three layers have been defined according to several objectives, the most prominent of them are stated below:

a) **Separate mechanisms from policies:** this objective aims to add flexibility and simplicity to the tool by completely separating the mechanisms required to monitor a particular security policy from its specifications.

b) **Scalable monitoring:** due to the huge increase of network bandwidths nowadays, NIDSs need to be able to monitor this huge amount of network traffic in the most efficient way without missing any potential threat.

c) **Attacks resistance:** one of the ways for attackers to achieve their goals successfully is to first disable the NIDS before targeting their victims and therefore securing this NIDS is of paramount importance.

2.1.2.3 Packet Capture

In this layer, a stream of packets are extracted from the network stream and passed to the event engine layer. In order to do so, Bro uses *libpcap* library which offers support for most network technologies and protocols. *Libpcap* works by using efficient and powerful expression filtering in order to reduce the amount of packets to analyze. As an example of this, if one wants to not capture SMB related network traffic (port 445); *libpcap* can be configured to ignore all related traffic to port number 445. Using this concept, Bro enables security experts to dynamically filter network traffic during run-time according to their policy requirements.

2.1.2.4 Policy-neutral Event Engine

This layer represents the core of Bro NIDS and its job is to analyze all network packets received from the packet capture layer. By getting these raw packets, it sorts them chronoligically, and decodes application layer protocols. This layer generates at the end of this process events based on the policy layer requirements and has to be very scalable and attack resistant. The event engine layer consists of several types of analyzers where each of them is designed for a well-defined task (decoding a network protocol, signature-matching, etc...). The event engine layer is devided into four major components:

- **State Management:** in Bro NIDS, each packet belongs to one single *connection*. Bro defines its own connection state expiration mechanism, this is due to the fact that on large network volumes there exists a lot of *crud* connections; network packets that show connections have been ended from both ends (FIN packet) but do not conform to any pre-defined standard. One common explanation for this type of packets are link-level errors and network protocols implementations persistant bugs.
- **Transport Layer Analyzers:** Bro TCP analyzer keeps track of the various state changes in each connection, keeps track of acknowledgements sent, handles retransmissions and does much more. This results in a real byte stream of payload data and fixes some drawbacks related to Snort IDS.
- **Application Layer Analyzers:** Bro also analyzes diffrent application layer protocols such as: HTTP, SMTP, or DNS.
- **Infrastructure:** includes components like event and time management, Bro scripting language interpreter, and other data structures.

2.1.2.5 Policy Layer

This is where security experts/analysts define their environment specific network security policies. This is achieved by writing handlers for each event raised by the event engine layer and which defines the constraints/alerts to generate for each relevant event. These event handlers are coded using Bro's own scripting language which has been designed according to network intrusion detection paradigm, proving a set of pre-defined functionalities, like: dynamic memory management, dynamic typing, regular expressions. During Bro startup, all the scripts enabled by default by the user are loaded and thus only the needed analyzers are loaded leading to less resource consumption. Last but not least, Bro comes with some pre-defined policy scripts that perform a wide range of tasks, but security experts can extend it with their own policy scripts and enabling them at startup.

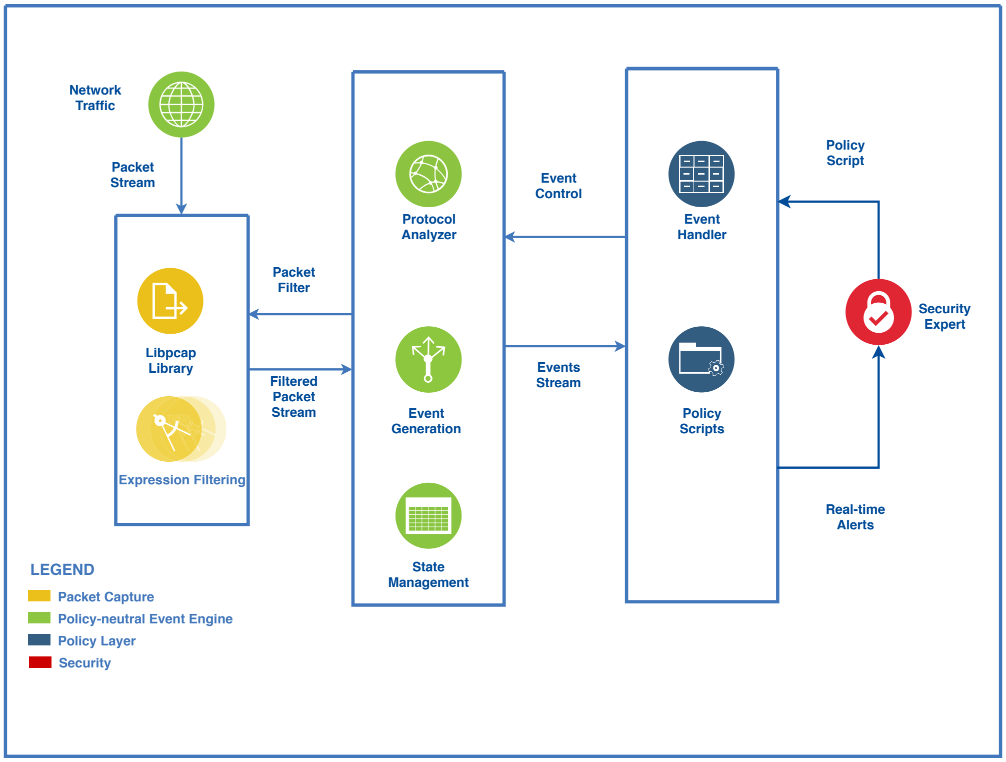


Figure 2.4: Bro NIDS Architecture [76].

2.1.2.6 Log Files

After analyzing this stream of network packets and applying the required analyzers and policy scripts, Bro generates diffrent types of log files depending on the analyzer triggered and the network protocols detected, as shown in Figure 2.5. Consequently, Bro NIDS log files are categorized into six categories [76, 1] that are presented below:

2.1.2.7 Network Protocols

This category gathers all the Bro log files that are related to the most common network protocols. Each time a protocol which analyzer is enabled at startup is detected during the analysis step, Bro generates automatically a log file for that specific protcol and stores inside it all the information that it deems necessary to be logged. The most important log files found in this category are: *a*) *conn log:* which contains all the information about any Internet connection that Bro considered as essential including IP addresses and ports, cooonection state and history, number of bytes and packets sent between both ends (originator and responder IPs). *b*) *http log:* which is generated automatically if Bro finds any connection that is requiring the usage of HTTP protocol (even HTTPS is considered by Bro as an HTTP event), and contains in addition to conn log file features, more information that Bro extracts from the packet like: user-agent, HTTP response codes, HTTP request type, and so on. *c*) *dns log:* generated automatically if Bro finds any connection using DNS protocol and besides the features of the connection, it additionally contains information about the DNS query, DNS response, and so on.

2.1.2.8 Files

This category gathers all the Bro log files that are related to files transmitted over the network wire and detected in the packets payloads. This information varies from files names to files sizes and from MIME types to files overflow bytes. In addition to that, Bro offers the capability of generating MD5 and Sha1 hashes for these files; this can be useful as some malicious files can be checked using their respective hashes against benchmark online malware databases.

2.1.2.9 NetControl

This category is related to a Bro plugin-based framework that enables network traffic control to Bro's sensors, and everything is allowed by default.

2.1.2.10 Detection

This category contains all the Bro logs files generated after a detection of an anomaly from an external source matching (intel log and signature log), or if some error is detected on the network traffic like SSL certificate errors for example (notice log).

2.1.2.11 Network Observations

This category contain Bro log files that vary from SSL certificates (known_certs log) to hosts that have completed TCP handshakes (known_hosts log), and from knwon services (known_services log) to software information being used on the network (software log).

2.1.2.12 Other Log Files

This category is related to detected failures in the *Dynamic Protocol Detection (DPD)* (dpd log), but also to unexpected network traffic activity detected by Bro (weird log).

Finally, we explore in more details the features of interest within Bro connection log (conn log) [76, 1]; since our proposed work/approach uses this log file as input. Figure 2.6 shows these features of interest. In Bro NIDS, the word *Originator* refers to the host that requested a connection initiation or initiated the Internet connection, whereas, the word *Responder* refers to the network host that this originator wants to communicate with. each of these two ends have four features of interest to our work alongside with other features that will not be discussed here. *a*) *originator IP*: represents the IP address of the host that is initiating the Internet connection, *b*) *originator port*: represents the TCP/UPD port of the network service that the originator requested, *responder IP*: represents the IP address of the responding host, *responder port*: represents the TCP/UDP port that this host used to reply to the originator's request, *originator bytes*: represents the total amount of bytes sent by the originator on the wire, *responder bytes*: represents the total amount of bytes sent in the responder's reply, *originator packets*: represents the number of packets sent by the originator to the responder on the wire, and *responder packets*: represents the number of packets sent in the responder's reply.

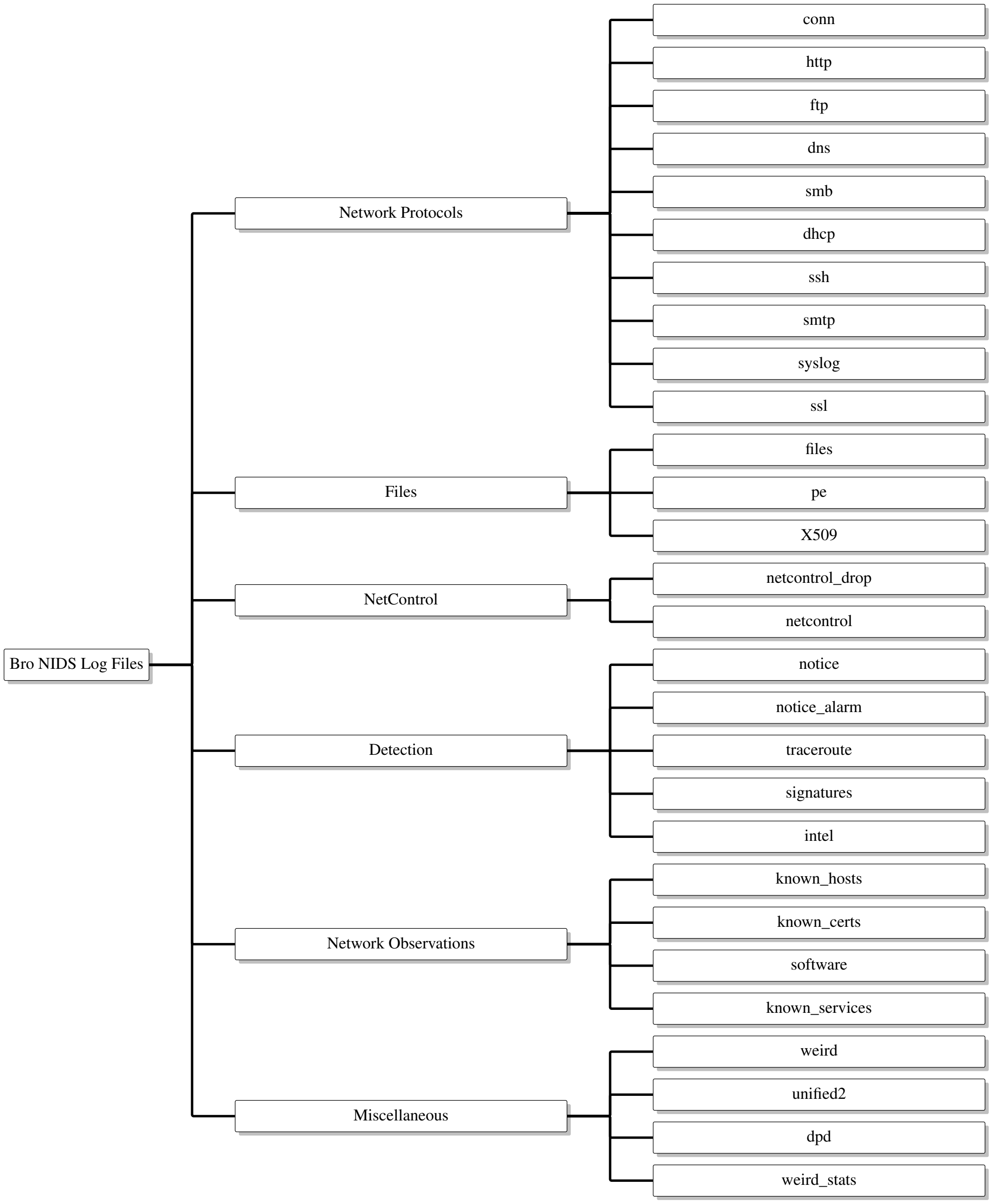


Figure 2.5: Bro NIDS Generated Log Files Taxonomy [76, 1].

2.2 Anomaly Detection Existing Approaches

According to our litterature review on network anomaly field, and more precisely to these two surveys [10, 45], network anomaly detection proposed techniques can be splitted into three categories: *machine learning* techniques, *statistical* techniques, and *soft-computing* techniques (Figure . 2.7). *Machine learning*

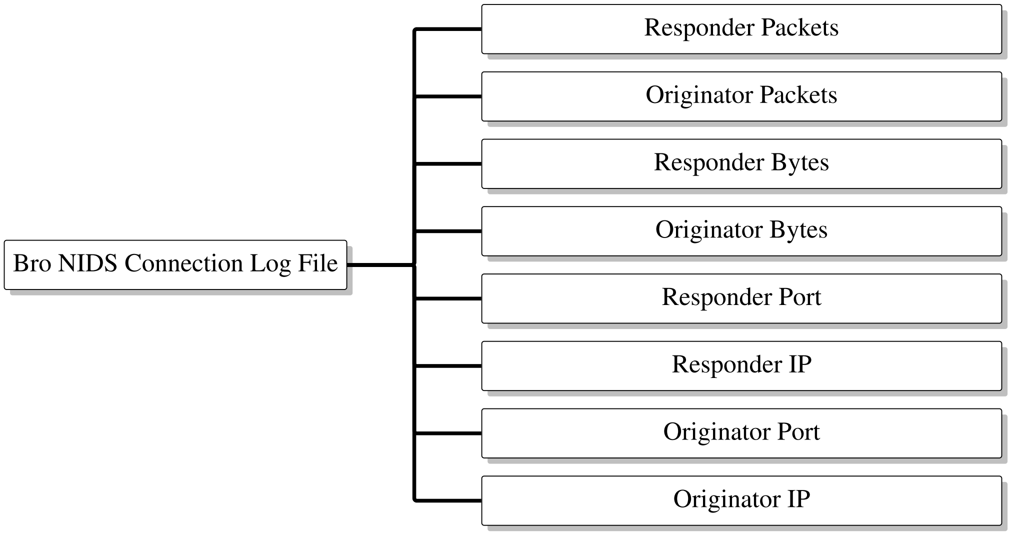


Figure 2.6: Bro NIDS Generated Connection Log Files Features [76, 1].

techniques are splitted into two main categories: *supervised* or *unsupervised* techniques. *Statistical* techniques which mainly depend on time-series analysis in order to detect abnormal activites (peaks) according to a pre-computed threshold, and *soft-computing* techniques which are better when it comes to deal with an environment with uncertainties and unprecisions.

2.2.1 Machine Learning Anomaly Detection

Machine learning based techniques use a set of common and most-known machine learning approaches and methodologies in order to detect abnormal activity on networks' traffics. As shown in Figure 2.8, this category of network anomaly detection approaches is devided into two sub-categories: *a*) **supervised** techniques which require generally a *training* phase in which a model is trained and built from a training data, and a *testing* phase which allows to test the accuracy of the generated model against testing data. The data used for these models is generally labelled and the goal of the built and validated model is to predict future unlabelled data targeting a certain class of labels. *b*) **Unsupervised** techniques on the other hand do not require to train a model and use generally unlabelled datasets, one of the most common machine learning techniques used in this sub-category are *clustering* algorithms.

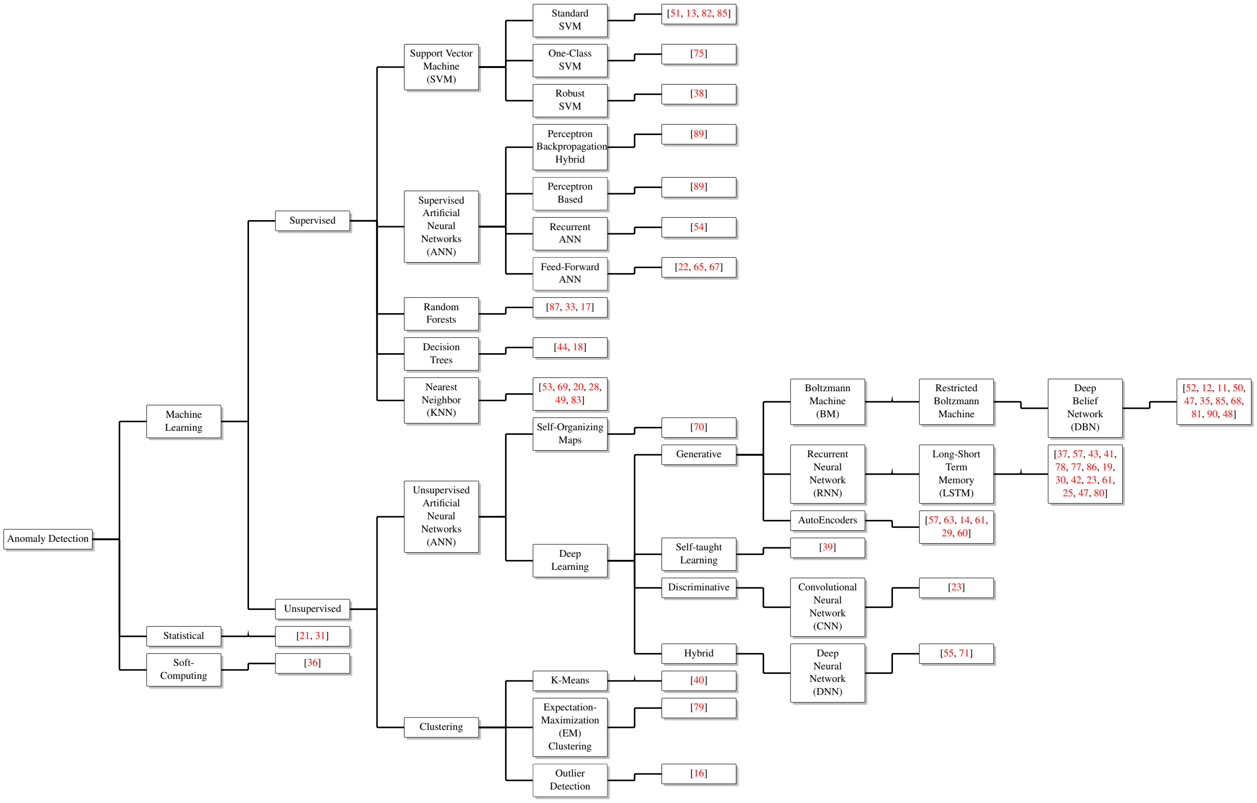


Figure 2.7: General Overview of Anomaly Detection Approaches. [10, 45]

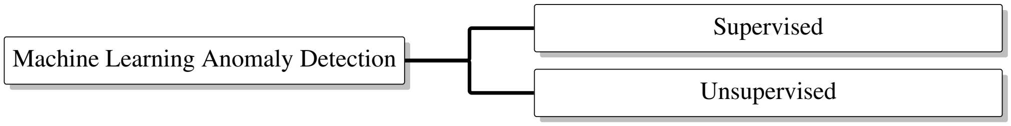


Figure 2.8: Classification of Machine Learning Anomaly Detection Approaches. [10, 45]

2.2.1.1 Supervised Machine Learning Anomaly Detection

Supervised-based anomaly detection techniques are very commonly used since they often require a pre-trained accurate model and these techniques in most of the time are accurate enough and give good classification and prediction results. However, these techniques suffer from some limitations:

a) Require generally several iterations of training and testing phases in order to generate the best accurate models,

b) Delay in the deliverance time of analytics results especially when it comes to critical data or projects that need to be delivered within very fast time delays.

This family of machine learning can also be devided into several sub-categories according to our litterature review[10, 45] as shown in Figure 2.9.

More precisely, this category is devided into four sub-categories:

a) **support vector machine (SVM)** based network anomaly detection techniques in which the algorithm generally require data points and outputs the resulting *hyperplane* (a simple separating line in a two-dimension space) that best defines the optimal decision boundary that separates the diffrent targeted classification class label's values. In other words, the data points are grouped on the sides of this *hyperplane* and each group is considered as an independant class.

b) **Random forest** based network anomaly detection technique on the other hand belong to the family of *ensemble methods* and are used to build predictive models for both classification and prediction problems. The model built using random forest algorithms creates a forest representation of random uncorrelated decision trees in order to take the best classification or prediction decision at the end.

c) **Decision trees** on the other side are quite similar to the previous one (random forests); to be more precise, decision trees are built on the entire dataset using all the features of interest to the targeted problem, whereas random forests randomly select observations and specific features to construct multiple decision trees and computes an average metric on the results. More precisely, each tree contributes with its own *vote*, and the class that receives the most votes by majority ruling is considered as the result of the classification/prediction.

d) **Supervised artificial neural networks (SANN)** take input vectors in the network and use them to produce output vectors, these output vectors are then fed to the algorithm which will compare them with a set of targeted/desired vectors. If an error is detected when there is a considerable diffrence between the resulting ouputs and the desired ones. This is repeated and each time the weights of the network are adjusted based on these error values, until the resulting outputs and the desired ones match.

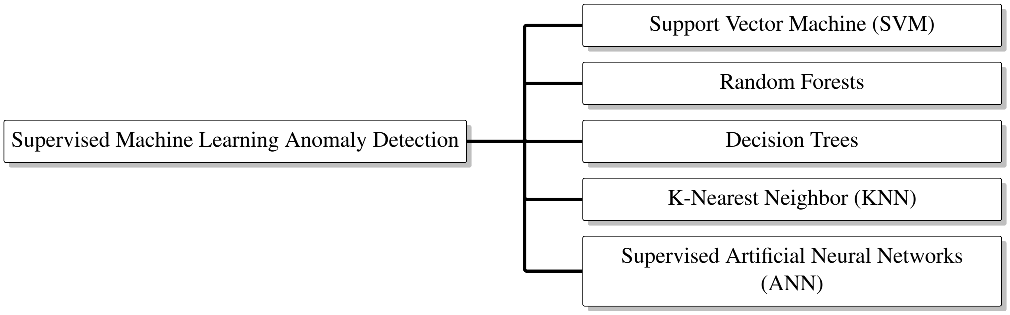

Supervised Machine Learning Anomaly Detection
Support Vector Machine (SVM)
Random Forests
Decision Trees
K-Nearest Neighbor (KNN)
Supervised Artificial Neural Networks (ANN)

Figure 2.9: Classification of Supervised Machine Learning Anomaly Detection Approaches. [10, 45]

2.2.1.2 Support Vector Machine Anomaly Detection

Suppor vector machine network anomaly detection techniques are devided according to the litterature review [10, 45] into three main classes, as shown in Figure 2.10.

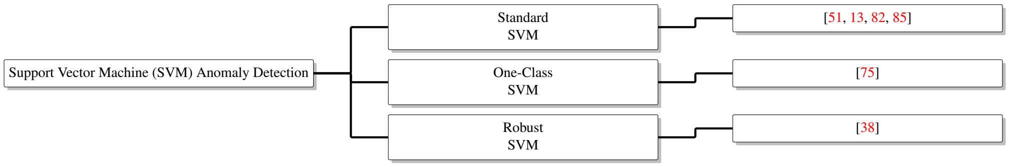


Figure 2.10: Classification of Support Vector Machine Anomaly Detection Approaches. [10, 45]

a) ***Standard SVM:*** four prominent works fall into this category. In [51], the authors used gradual feature removal technique in order to reduce the number of features used to 19 critical ones chosen from KDDCUP99 [64]. Then, using a combination of clustering techniques, ant colony algorithm, and support vector machines (SVM), they built an efficient classifier to detect normal and abnormal network traffic. At the end of their work, they evaluated their classifier and showed that it achieves an accuracy of 98.62% using 10-fold cross validation and an average *Matthews* correlation coefficient (MCC) of 86.11%. In [13], the authors proposed *mutual information-based feature selection method* for intrusion detection systems. More precisely, two feature selection methods are proposed and their performance is comapred to a mutual information-based feature seletion method. This comparison is achieved using both *linear* (linear correlation coefficient) and *non-linear* (mutual information). Furthermore, they proposed an intrusion detection algorithm using their own improved *least squres support vector machines*. At the end of their work, they conducted their experiments on KDDCUP1999 [64] dataset and after evaluation and validation, their proposed feature selection algorithm achieved the highest accuracy results; with an average of 99.8% on normal traffic, 99% on denial of service (DoS) traffic, 99.83% on probing attacks traffic, 99.91% on remote to login (R2L) traffic, and 93.16% on user to remote (U2R) traffic compared with other techniques. In [82], a support vector machine anomaly detection based technique was proposed by authors which analyses tremendous amounts of *NetFlow* traffic. Their proposed approach uses a special kernel function which considers both contextual and quantitative features select from *NetFlow* records. The following Table 2.2 best shows these accuracy results on diffrent attacks:

Table 2.2: Accuracy results achieved in [82]

Attack Type	Accuracy	False Positives Rate	True Negatives Rate
Nachi Scan	89.6%	0.4%	99.6%
Netbios Scan	93.8%	0%	100%
Popup Scan	91.5%	2.3%	97%
SSH Scan & TCP Flood	91.7%	1.1%	98.9%
DDoS UDP Flood	91.5%	2.2%	97.8%
DDoS TCP Flood	90.7%	3.3%	96.7%
Stealthy DDoS UDP Flood	93.8%	0%	100%
DDoS UDP Flood & Traffic Deletion	93.4%	0%	100%

In [85], a novel algorithm called *support vector machine based on the restricted Boltzmann machine (SVM-RBM)* is proposed in order to detect network anomalies. Restricted Boltzmann machine (RBM) are used during the feature extraction step and choose the best gradient descent algorithm using Spark in order to train the support vector machine (SVM) classifier. Furthermore, in order to improve the performance of SVM-RBM, diffrent numbers of hidden units are tested and it was shown that the SVM classification highly depends on the learning capability of these features using RBM.

b) ***<u>One-Class SVM:</u>*** the most prominent work that has been proposed in this category of SVMs is the work proposed in [75]. The authors proposed a hybrid machine learning approach in order to detect network anomalies. More precisely, they used *self-organized feature map (SOFM)* in order to use SVM without previous knowledge. Then, *passive TCP/IP Fingerprinting (PTF)* is used in order to reject incomplete network traffic that is considered violating TCP/IP standards. After that, a feature selection technique based on *genetic algoritm* is used in order to extract optimal information from raw packets. They also took into account the temporal relationships between data during the preprocessing step and fed them to their *Enhanced SVM*. Finally, they experimented their technique alongside with other techniques on both MIT Lincoln Labs dataset alongside with real captured data from real network traffic, and the validation was achieved using *m-fold* cross-validation. The following Table 2.3 shows the results of their evaluation:

Table 2.3: Accuracy results achieved in [75]

Approach	Kernel Function	Detection Rate	False Positives Rate	False Negatives Rate
Soft Margin SVM	Inner Product	90.13%	10.55%	4.36%
Soft Margin SVM	Polynomial	91.10%	5%	10.45%
Soft Margin SVM	RBF	98.65%	2.55%	11.09%
Soft Margin SVM	Sigmoid	95.03%	3.9%	12.73%
Soft Margin SVM	Average	93.73%	5.5%	9.66%
Soft Margin SVM	Standard Deviation	3.91%	3.51%	3.66%
One-Class SVM	Inner Product	53.41%	48%	36%
One-Class SVM	Polynomial	54.06%	45%	46%
One-Class SVM	RBF	94.65%	20.45%	44%
One-Class SVM	Average	67.37%	37.82%	42%
One-Class SVM	Standard Deviation	23.62%	15.11%	5.29%
Enhanced SVM	Sigmoid	87.74%	10.2%	27.27%

c) ***Robust SVM:*** in [38], a comparative study was realised between the performance of *robust support vector machines (RSVMs)*, *conventional/standard SVMs*, and *nearest neighbor classifiers (KNNs)* on KDDCUP99 [64] dataset. The results of their evaluation show that RSVMs outperform the two other techniques achieving higher accuracy results with lower false positives rates. Table 2.4 shows these results:

Table 2.4: Accuracy results achieved in [38]

Approach	Attack Detection Rate	False Positives Rate
RSVMs	81.8%	Less than 1%
SVMs	81.8%	Less than 1%
KNN	63.6%	Less than 1%

2.2.1.3 Random Forests Anomaly Detection

As shown in Figure 2.11, there are mainly three important noticeable works using random forests in order to detect network anomalies. In [87], a random forest network anomaly detection technique was proposed in order to detect misuse, and abnormal behaviors on IDS networks. For misuse detection, intrusion patterns are represented by the random forest automatically over the training data. Then, intrusions are detected by matching network activities against the trained/detected patterns. As for anomalies, outlier detection capability of random forests are used. This hybrid approach improves the detection performance by combining the advantages of both misuse and anomaly detection. Their approach was evaluated on KDDCUP99 [64] dataset and the results of the validation show that this technique outperforms other unsupervised techniques.

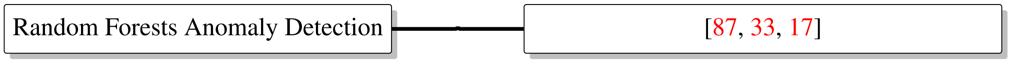


Figure 2.11: Classification of Random Forests Anomaly Detection Approaches. [10, 45]

In [33] on the other side, the authors conducted a comparative study between *supervised probabilistic* and *predictive* machine learning techniques for network anomaly detection techniques. More precisely, two probabilistic techniques *Naive Bayes* and *Gaussian* were compared with two other predictive techniques *Decision trees* and *Random forests*. For their evaluation and experiments, KDDCUP99 [64] dataset was used and the ability of these four techniques to detect four types of attacks was compared. The results of their comparative study are shown in Table 2.5 :

In [17], a large scalable botnet detection tool called *Disclosure* was proposed. They selected several features that are considered important from *NetFlow* data in order to detect reliably botnets' *command and control (C&C) centers* on benign *NetFlow* traffic. The evaluation of their approach was conducted on two large real network traffic datasets (billions of records per day) and proved that *Disclosure* can detect botnet C&C channels during very fast time delays. Table 2.6 shows the details about both real large networks used for the evaluation, and Table 2.7 shows the results of their approach.

Table 2.5: Detection rates achieved in [33]

Approach	Population Size	DoS	Probe	R2L	U2R
Gaussian	8020	96.7%	87.3%	13.6%	48.6%
Gaussian	8416	96.7%	87.4%	13.6%	48.6%
Gaussian	8812	84.3%	86%	13.5%	48.6%
Naive Bayes	8020	79.1%	81.6%	12.5%	84.3%
Naive Bayes	8416	79.1%	81.4%	12.5%	81.4%
Naive Bayes	8812	79.1%	81.3%	12.4%	84.3%
Decision Tree	8020	97.2%	73.6%	9.3%	28.6%
Decision Tree	8416	96.8%	77.9%	7.4%	24.3%
Decision Tree	8812	96.6%	74.7%	9%	30%
Random Forest	8020	97.2%	74.5%	5.5%	25.7%
Random Forest	8416	97.1%	75.8%	4.8%	34.3%
Random Forest	8812	97.1%	77.7%	6%	25.7%

Table 2.6: Experimental networks details used in [17]

Network	Sampling	Flows Per Day	Unique IP Addresses
Inter-University Network (N1)	1:1	1.2 billion	28 million
Tier 1 ISP (N2)	1:10:000	400 million	50 million

Table 2.7: Experimental Results Achieved in [17]

False Positives Rates (Point on the ROC curve)	Servers Flagged as C&C From N1	Servers Flagged as C&C From N2
1%	12,383	4,937
0.5%	7,856	3,166
1%	6,295	1,958
1%	132	960

2.2.1.4 Decision Trees Machine Anomaly Detection

There are two major works that fall into this class of network anomaly detection techniques. In [44], an approach using decision trees in order the process of matching used by signature-based techniques is deployed combined with a set of machine learning clustering techniques. Experimental evaluation of the approach proves that it detects anomalies in a fast manner compared to *Snort*'s detection module.

In [18], *Exposure*, a large scalable system that detects malicious domains using *passive DNS analysis* *techniques* was proposed. A total set of 15 features was used and extracted from DNS traffic. The experimental evaluation was conducted on a real-world dataset of 100 billion DNS requests, and a real-life deployement over two weeks in an ISP, and proved the scalability of their approach alongside with its high capability of detecting malicious domains used in a variety of malicious activities. Table 2.8 shows the accuracy results of their approach:

Table 2.8: Experimental Results Achieved in [18]

Validation Method	Area Under the Curve (AUC)	Detection Rate	False Positives Rate
Full Data	99.9%	99.5%	0.3%
10-folds Cross-Validation	98.7%	98.5%	0.9%
66% Percentage Split	98.7%	98.4%	1.1%

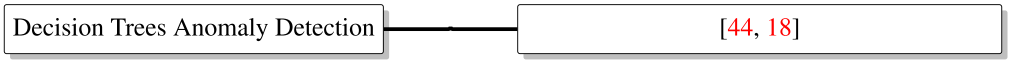


Figure 2.12: Classification of Decision Trees Anomaly Detection Approaches. [10, 45]

2.2.1.5 K-Nearest Neighbor Anomaly Detection

Five prominent research works have been proposed that are based on K-Nearest Neighbor in order to detect network anomalies. In [53], an approach based on KNN classifier is proposed in order to distinguish between normal and abnormal (intrusive) programs. This approach works by representing programs behaviors into system calls frequencies, where each system call is considered as a word and the set of all system calls during one program's execution are considered as one document. Then KNN is used to classify these documents since this method has proven its advantages in the past when it comes to deal with text categorisation. The evaluation of this approach was conducted on KDDCUP99 [64] dataset and proved that it can detect in an effictive way intrusive attacks achieving lower false positives rates. Table 2.9 show the results of this evaluation:

In [69], an approach for anomaly detection was proposed by the authors which used a distributed, clustering-based algorithm. The approach was evaluated using a simulated environment using sensor (multiple sensors) collected data and demonstrated considerable accuracy results compared to a centralised sensor scheme.

Table 2.9: Experimental Results Achieved in [53]

Attack Type	Instances	Detected	Detection Rate
Known Attacks	16	16	100%
Zero-Day Attacks	8	6	75%
Both Attacks	24	22	91.7%

In [20], a novel method was introduced which uses first *principal component analysis (PCA)* which projects data elements onto a vector space in order to represent the diffrent variations in those data elemts. Then, a comparative study is conducted between *decision tree* and *KNN* algorithms. The experiments were conducted on KDDCUP99 [64] dataset and shows that KNN algorithm outperforms decision trees, as shown in Table 2.10, and in Table 2.11:

Table 2.10: Confusion Matrix achieved by using KNN with PCA in [20]

Predicted As	Population Size	Benign	Probing	DoS	U2R	R2L
Normal	60593	99.5%	0.27%	0.23%	0%	0%
Probing	4166	13.87%	74.4%	11.37%	0%	0.36%
DoS	229853	2.68%	0.18%	97.14%	0%	0%
U2R	228	35.96%	14.47%	39.03%	7.91%	2.63%
R2L	16189	97.49%	1.71%	0%	0%	0.8%

Table 2.11: Confusion Matrix achieved by using Decision Tree algorithm with PCA in [20]

Predicted As	Population Size	Benign	Probing	DoS	U2R	R2L
Normal	60593	99%	0.85%	0.12%	0%	0.03%
Probing	4166	29.6%	66.8%	3.5%	0.10%	0%
DoS	229853	2.42%	0.33%	97.25%	0%	0%
U2R	228	92.98%	0%	0.44%	6.58%	0%
R2L	16189	99.94%	0%	0.06%	0%	0.01%

In [49], a novel supervised network anomaly detection technique based on *transductive confidence machines for k-nearest neighbors (TCM-KNN)* was proposed. This approach is able to effectively detect anomalies with higher detection rates, lower false positives rates by using lesser data and selected features compared to the classical approaches. The evaluation that was conducted on KDDCUP99 [64] dataset showed that this approach outperformed other state of the art techniques, as shown in Table 2.12.

Table 2.12: Experimental Results Achieved in [49]

Approach	True Positives Rates	False Positives Rates
SVM	98.7%	2.7%
Neural Network	98.3%	2.2%
KNN	97.7%	4.8%
TCM-KNN	99.6%	0.1%

Last but not least, in [83], *payload-based anomaly detection (PAYL)* is proposed which analyses the payload of network traffic in a fully automatic way. During the training step, the profile of the network payload is built as a byte frequency representation using also standard deviation of the payload flows to each single network host and its associated port number. Then KNN is used to classify and detect anomalies using *Mahalanobis* distance in order to compute the similarity between new data and the pre-computed payload profile. The performance of their approach was demonstrated using KDDCUP99 [64] dataset and also a real network traffic collected on the *Columbia Computer Science (CS) department* network. The results of their evaluation on both these datasets achieve high accuracy rates; with almost 100% accuracy and 0.1% false positives rates when it comes to HTTP port (80) network traffic.

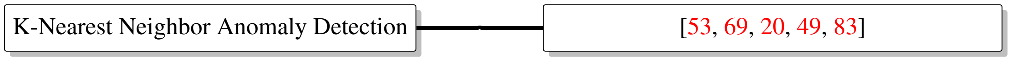


Figure 2.13: Classification of K-Nearest Neighbor Anomaly Detection Approaches. [10, 45]

2.2.1.6 Supervised Artificial Neural Networks Anomaly Detection

Supervised artificial neural networks (SANN) on the other hand are categorized according to our litterature review [10, 45] into three main categories, as shown in Figure 2.14:

a) **Perceptron Backpropagation Hybrid:** One major work has been noticed using this technique: [89], in which the authors introduced *hiearchical intrusion detection (HIDE)* which is a tool that detects network traffic anomalies and attacks using a combination of statistical preprocessing techniques and neural network classification algorithms. Moreover, they tested five distinct neural network

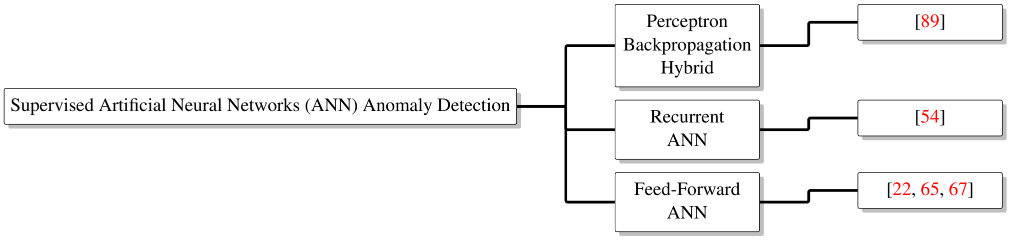

Supervised Artificial Neural Networks (ANN) Anomaly Detection
Perceptron Backpropagation Hybrid
[89]
Recurrent ANN
[54]
Feed-Forward ANN
[22, 65, 67]

Figure 2.14: Classification of Supervised Artificial Neural Networks Anomaly Detection Approaches. [10, 45]

algorithms: *Perceptron*, *Backpropagation (BP)*, *Perceptron Backpropagation Hybrid (PBH)*, *Fuzzy ARTMAP*, and *Radial-based Function*. The results of their comparative study using these five algorithms showed that both BP and BPH outperform the other three algorithms, and that they both can effectively detect *UDP flooding* attacks.

b) **Recurrent ANNs:** In [54], proposed an approach that uses recurrent ANNs in order to reduce and fix the problem of high false positives rates in the *signature-based* intrusion detection techniques. In order to do so and improve the performance of their system, they used a combination of both discriminative training and generic keywords representations. Their evaluation and validation was conducted on KDDCUP99 [64] dataset and achieved 80% of detection rate with approximately one false alarm (false positive) per day.

c) **Feed-Forward ANNs:** in [22], a feed-forward ANN based misuse detection system was proposed in order to overcome one major flaw/drawback of rule-based techniques which consists of missing detection when it comes to deal with patterns completely diffrent from the expected ones. Their evaluation was conducted by computing both *root mean square error (RMSE)* and *correlation* metrics and are shown in Table 2.13.

Table 2.13: Experimental Results Achieved in [22]

Training Data RMSE	Test Data RMSE	Training Data Correlation	Test Data Correlation
0.058298	0.069929	0.982333	0.975569

In [65], the authors examined two systems for network anomaly detection using respectively feed-forward and perceptrons for the first one, and self-organizing maps for the second system. These two systems were evaluated on KDDCUP99 [64] dataset and showed that the learning capability of neural networks can help improve the detection of abnormal network activity (DoS, Port Scan, DDoS, and doorknob attacks), as shown in Table 2.14.

Table 2.14: Experimental Results Achieved in [65]

	Correct Normal Predictions	False Negatives Rates	Correct Attack Predictions	False Positives Rates
Union of All Attacks	100%	0%	24%	76%
DoS	100%	0%	24%	76%

Last but not least, in [67], a neural network based network anomaly detection approach was proposed using feed-forward alongside with backpropagation technique in order to train the model on 10% of KDDCUP99 [64] dataset. The evaluation results showed that the proposed algorithm detects diffrent attack types in an effective fashion with less false positives and false negatives rates, compared to some state of the art techniques. These results are shown in Table 2.15.

Table 2.15: Experimental Results Achieved in [67]

Approach	Accuracy	False Positives Rates	False Negatives Rates	Number of Epochs
Proposed Approach	94.93%	0.002%	0.7%	1067
Standard ANNs	87.07%	6.66%	6.27%	412

2.2.1.7 Unsupervised Machine Learning Anomaly Detection

Unsupervised machine learning anomaly detection techniques on the other hand are splitted based on our litterature review and some available surveys on the topic [10, 45] into two main categories: *a*) *Unsupervised Artificial Neural Networks (UANN):* where ANNs are used to detect network anomalies in a completely unsupervised fashion without requiring training and testing phases. *b*) *Clustering:* based network anomaly detection techniques on the other hand are also unsupervised and work in a way of clustering network data into diffrent clusters and then make a decision on which clusters' points to consider as anomalous or not based on diffrent criterias. Figure 2.15 shows these two categories:

2.2.1.8 Unsupervised Artificial Neural Networks

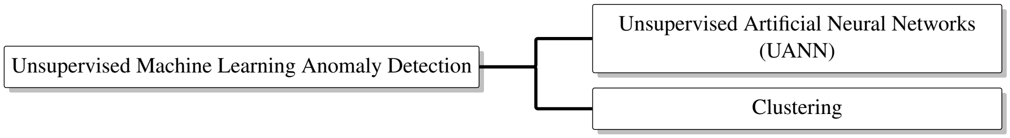

Unsupervised Machine Learning Anomaly Detection
Unsupervised Artificial Neural Networks (UANN)
Clustering

Figure 2.15: Classification of Unsupervised Machine Learning Anomaly Detection Approaches. [10, 45]

For unsupervised artificial neural networks based network anomaly detection techniques, they can be splitted into two main categories: *a*) *self-organizing maps:* are used to cluster and visualize network data extensively, whereas *b*) *deep learning:* techniques work in a way to mimic the human brain where each neuron is represented as a node and these nodes are connected with other through multiple layers; inputs layer, hidden layers, and outputs layer. Figure 2.16 shows these two categories.

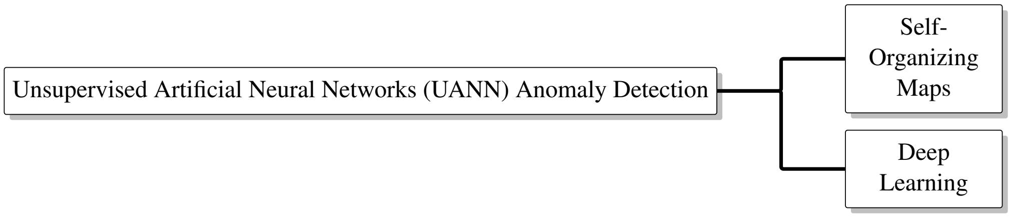

Unsupervised Artificial Neural Networks (UANN) Anomaly Detection
Self-Organizing Maps
Deep Learning

Figure 2.16: Classification of Unsupervised Artificial Neural Networks Anomaly Detection Approaches [10, 45]

- **Self-Organizing Maps:** As shown in Figure 2.17, this sub-category of unsupervised artificial neural networks based network anomaly detection techniques has a noticeable major work [70], where the authors introduced *integrated network-based Ohio University network detection service (INBOUNDS)* as a network based intrusion detection system. This tool contains a module called *anomalous network-traffic detection with self-organizing maps (ANDSOM)* where self-organized maps were deployed. In more details, *ANDSOM* represents each network connection by extracting six features of interest into a six-dimensional vector. Then, it creates a two-dimensional *lattice* of *neurons* for each network service. After that, and during real-time running of the tool, each coming new network connection is fed to *ANDSOM* which attributes its respective self-organized map (SOM), and a distance is computed between this connection SOM and the closest neuron; if this distance is above a fixed threshold, then it is considered as an anomalous network connection. The results of their experimentation and evaluation show that the percentage of false positives for both DNS and HTTP traffic is of 1.18% and 1.16% respectively.

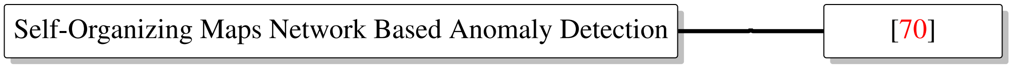


Figure 2.17: Self-Organizing Maps Network Based Anomaly Detection Publications. [10, 45]

- **Deep Learning:** As shown in Figure 2.18, deep learning network based anomaly detection techniques are on the other hand categorized into diffrent sub-categories.

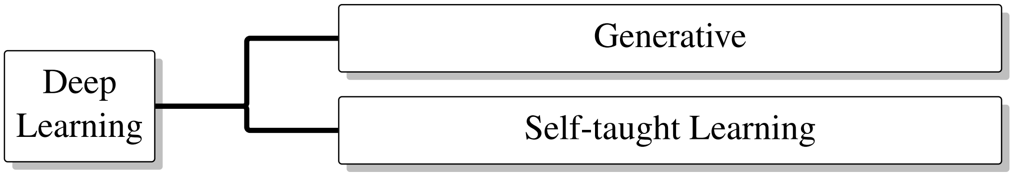


Figure 2.18: Classification of Deep Learning Based network Anomaly Detection Approaches. [10, 45]

  a) **Generative:** This sub-category is in turn also splitted into three main families, as shown in Figure 2.19.

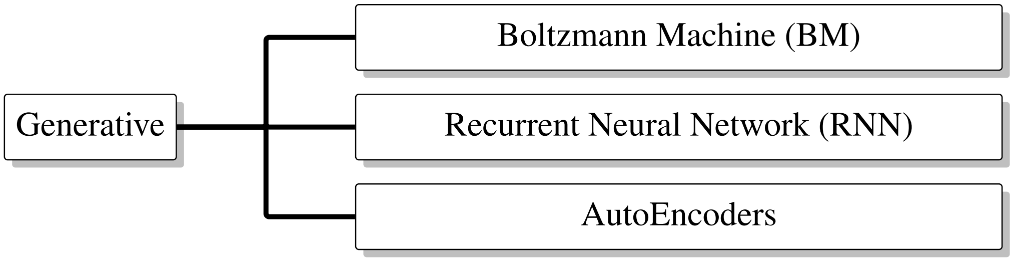


Figure 2.19: Classification of Generative Deep Learning Network Based Anomaly Detection Approaches. [10, 45]

  - **Boltzmann Machine (BM):** The major works that fall into this category of network anomaly detection techniques mostly are sub-categorized as *restricted Boltzmann Machine (RBM)* and in turn to *deep belief network (DBN)* based techniques, as shown in Figure 2.20. There are a lot of major works that have been proposed in the past using Boltzmann Machine (BM) approach; ten of them are considered as important and thus shall be discussed next.

    In [52], a hybrid network anomaly detection technique was introduced which combined both *autoencoders* and *restricted Boltzmann Machines (RBMs)*. First, the authors used

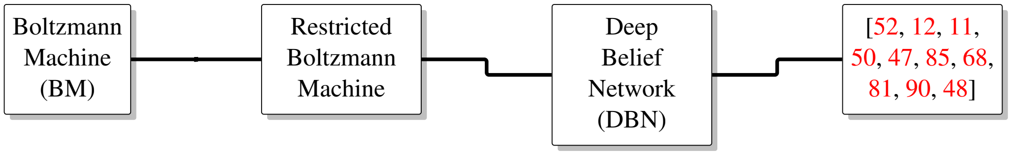


Figure 2.20: Classification of Boltzmann Machine (BM) Network Anomaly Detection Approaches. [10, 45]

autoencodes as a dimensionality reduction technique in order to simplify the data representation and improve the performance of their algorithm and extract the most important features. Then, RBMs are used in order to detect malicious code; where the output vector of the last layer of the RBM is fed as input vector for a *backpropagation* neural network. The experimentation and evaluation of their proposed approach showed that the detection accuracy of this hybrid approach outperforms other systems where only a single DBN is used. These results are depicted in the following Table 2.16:

Table 2.16: Experimental Results Achieved in [52]

Model Used	True Positives Rate	False Positives Rates	Accuray	CPU Time (Seconds)
DBN	95.34%	9.02%	91.4%	1.126
AutoEncoders & DBN (5 Training Iterations and 5 Fine-Tuning Iterations)	96.79%	15.79%	89.75%	2.625
AutoEncoders & DBN (10 Training Iterations and 5 Fine-Tuning Iterations)	93.35%	9.17%	88.95%	1.147
AutoEncoders & DBN (10 Training Iterations and 10 Fine-Tuning Iterations)	92.2%	1.58%	92.1%	1.243

In [12], a novel approach of network anomaly detection using restricted Boltzmann machines (RBMs) and deep belief networks (DBNs) was proposed. It started by reducing the features dimensionality by using a one-layer RBM in a complete unsupervised fashion. Then, the resulting weights of this process are fed to another RBM which in turn produces a deep belief network (DBN). This second RBM uses a *fine-tuning* layer which consists of a *logistic regression (LR)* classifier using *multi-class soft-max*. The tool was developed and implemented using *C++* using *Microsoft Visual Studio 2013*. As for the experimentation and evaluation of the approach, KDDCUP99 [64] dataset was used for that purpose, and at the end of this process, the authors showed that this proposed technique outperforms other pre-existing deep learning techniques in terms of accuracy and detection speed; as shown in Table 2.17:

In [11], the authors demonstrated the usefullness of restricted Boltzmann machines (RBMs) in order to detect anomalous and normal *NetFlow* traffic. The evaluation of their work was

Table 2.17: Experimental Results Achieved in [12]

Method	Accuracy	True Positives Rates	CPU Time (Seconds)
[12]	97.9%	97.5%	8
[32] (4 Hidden Layers)	93.47%	92.33%	N/A
[52] (10 Training Iterations and 10 Fine-Tuning Iterations)	92.10%	92.20%	1.24

conducted on the benchmark dataset of UNBCIC [7].

In [50], the authors proposed a novel approach for detecting network anomalies on power grid's flow traffic. Their approach works by first by clustering the data into clusters based on their time occurrences, then based on the resulting clusters, determine which RBM to use. Then, by merging the most similar clusters, they choose an oveall RBM model and based on the pre-computed similarities between the real-time data and the used model, anomalies are detected. The results of their evaluation step show that their proposed techniques is more accurate in detecting anomalies on power grid's flow traffic compared to other techniques. These results are shown in Table 2.18:

Table 2.18: Experimental Results Achieved in [50]

Method	Number of Clusters	Average Convergence Tone Per Round (Seconds)	Average Accuracy
K-Means	4	32	73%
DBSCAN	10	37	82%
K-RBM [50]	17	45	91%

In [47], a hybrid method for network anomaly detection was proposed by the authors using a combination of *recurrent neural networks (RNNs)* and *restricted Boltzmann machines (RBMs)*. More precisely, *distributed embedding* technique was used to pre-process network data and make it suitable for the neural networks models. Then, an RBM model was used in order to extract feature vectors from network packets and an RNN model was used for the extraction of network flow features vectors. Finally, a soft-max layer was used which is fed with these flow vectors detecting anomalies in the result. Their evaluation was conducted on both UNBCIC [7] and KDDCUP99 [64] datasets and the results showed that their technique (RNN-RBM) achieves higher detection rates, recall, with lower false positives rates compared with other state of the art techniques. These results are depicted in the following Table 2.19:

In [85], a novel algorithm for network anomaly detection was introduced by the authors

Table 2.19: Experimental Results Achieved in [47]

Algorithm	Accuracy	Recall (DoS)	Recall (Probing)	Recall (R2L)	Recall (U2R)	Average Recall	Average FPR
KNN	49.2%	N/A	N/A	N/A	N/A	44.9%	46.46%
KNN-DS	37.4%	N/A	N/A	N/A	N/A	34.7%	35.29%
SVM	79.4%	81.1%	76.7%	21.4%	11.2%	N/A	0.6%
PLSSVM	99.8%	79%	87.6%	88%	18.4%	N/A	3.9%
ID3	N/A	99.9%	99.7%	99.5%	49.1%	N/A	0.43%
EID3	N/A	99.9%	99.8%	99.7%	99.8%	N/A	0.19%
Random Forest	91.4%	91.1%	55.1%	66.7%	100%	N/A	2.23%
Bayesian Network	90.6%	94.6%	83.8%	5.2%	30.3%	N/A	0.3%
Naive Bayes	78.3%	79.2%	94.8%	0.1%	12.2%	N/A	4.05%
MLP	N/A	96.9%	74.3%	0.3%	20.1%	N/A	0.53%
RNN-Based	95.2%	99.6%	59.7%	0%	38.1%	N/A	0.38%
RBF	N/A	N/A	N/A	N/A	N/A	90.7%	5.6%
RNN-RBM [47]	99.3%	99.4%	91.5%	97.7%	93.3%	N/A	0.02%

called *support vector machine based on the restricted Boltzmann machine (SVM-RBM)*. More precisely, RBMs were used in order to extract important features from the data and then *gradient descent* algorithm was used alongside with *Spark* in order to train the support vector machine (SVM) classifier. Moreover, they tuned their algorithm by using diffrent numbers of hidden layers in order to improve the performance of SVM-RBM. As for the evaluation of SBM-RBM, they depicted the *precision* results achieved using diffrent parameters compared with other existing algoritms: *Decision Tree (DT)*, *Naive Bayes (NB)*, and *Neural Network (NN)*.

In [68], the authors compared between three techniques in order to detect network anomalies on the KDDCUP99 [64] dataset; *support vector machines (SVMs)*, *backpropagation (BP)*, and *deep belief networks (DBNs)* using *restricted Boltzmann machines (RBMs)*. The results of this comparative study (both accuracy and time consumption) are depicted in the following Table 2.20:

Table 2.20: Experimental Results Achieved in [68]

Model	Time (Seconds)	Accuracy
Support Vector Machine (SVM)	13.53	91.36%
Backpropagation (BP)	16.85	89.07%
Restricted Boltzmann Machine (RBM)	11.65	95.25%

In [81], a hybrid approach for network anomaly detection was introduced by the authors called *Dynamic Recursive Deep Belief Neural Networks (DRDBNN)* which combined both *restricted Boltzmann machines (RBMs)* and *random forests*. In more details, they use RBMs first in order to extract the most important features to which they add a *time-varying factor* and a *forgetting coefficient* to improve them. Then, they feed these features to a random forest classifier which classifies them into normal or anomalous ones. The results of their evaluation are shown in the following Table 2.21:

Table 2.21: Experimental Results Achieved in [81]

Approach	Data Sampling Percentage	Detection Rate	False Positives Rate
DRDBNN [81]	10%	91.25%	4.85%
DRDBNN [81]	50%	92.27%	4.25%
DRDBNN [81]	80%	93.85%	4.06%
DBNs	10%	88.35%	5.84%
DBNs	50%	89.56%	5.39%
DBNs	80%	91.86%	4.76%
SVMs	10%	85.42%	5.77%
SVMs	50%	87.79%	5.86%
SVMs	80%	87.98%	5.08%
NNs	10%	83.56%	6.72%
NNs	50%	84.77%	6.09%
NNs	80%	86.93%	6.25%

In [90], in order to deal with the issue of redundant information, huge data size, and time-consuming training phases, a novel network anomaly detection approach was proposed using both *restricted Boltzmann machines (RBMs)* and *probabilistic neural networks (PNNs)*. First, by using the *nonlinear learning capability* of DBNs, features and attributes of importance are extracted. Then, in order to improve the training performance, *Particle Swarm Optimization (PSO)* algorithm is used to find the optimal number of nodes per hidden layer. Finally, PNNs are used in order to classify the results as anomalous or not. The evaluation of the work was conducted on KDDCUP99 [64] dataset and the results prove that this hybrid approach outperforms conventional techniques (PNN, PCA-PNN, and unoptimized DBN-PNN). These results are depicted in the following Table 2.22:

Table 2.22: Experimental Results Achieved in [90]

Approach	Running Time (Seconds)	Accuracy	Detection Rate	False Positives Rates
Unoptimized DBN-PNN	5.71	99.31%	91.75%	0.375%
Optimized DBN-PNN [90]	5.48	99.14%	93.25%	0.615%
PCA-PNN	6.16	98.28%	89%	1.33%
PNN	35.38	99.04%	89.25%	0.55%

Last but not least, and to conclude the most prominent works published in this sub-category, in [48], a novel approach for intrusion detection on *heavy-duty robots* was introduced which uses an algorithm called *improved deep belief networks (IDBNs)* and dynamic modeling. First, it had a security checking process which is able to detect targeted attacks from a specific cyber-domain. Then, a module for dynamic modeling and security detction is used to detect critical attacks that can cause physical damages.

– **Recurrent Neural Networks (RNNs):** The major works and publications that fall into recurrent neural networks based network anomaly detection category, as shown in Figure 2.21. There are twelve that are of interest and thus studied in more details in the following.

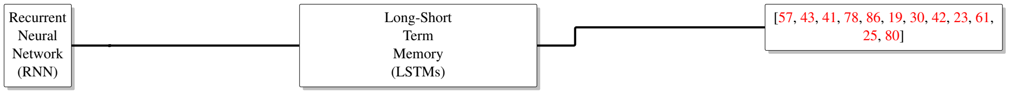

Recurrent Neural Network (RNN)
Long-Short Term Memory (LSTMs)
[57, 43, 41, 78, 86, 19, 30, 42, 23, 61, 25, 80]

Figure 2.21: LSTM-based Network Anomaly Detection Approaches. [10, 45]

In [57], a novel approach for anomaly detection called *bidirectional Long-Short Term Memory using Denoising Autoencoders (BLSTM-DAE)* where they process auditory spectral features. They use the reconstruction error obtained from autoencoders as an activation signal. This autoencoder is trained on a publicly available dataset which contains *in-home* situations like: watching televison scenarios, and playing scenarios. Their evaluation was conducted on more than 260 abnormal events and the results were compared with existing state of the art techniques, showing that this approach outperforms them. Table 2.23 shows these evaluation' results:

Table 2.23: Experimental Results Achieved in [57]

Approach	Precision	Recall	F1_Score
GMM	91.1%	87.8%	89.4%
HMM	94.1%	88.9%	91.4%
LSTM-CAE	91.7%	86.6%	89.1%
BLSTM-CAE	93.6%	89.2%	91.3%
LSTM-DAE	94.2%	90.6%	92.4%
BLSTM-DAE [57]	94.7%	92.0%	93.4%

In [43], a novel intrusion detection system (IDS) was proposed by the authors which applies the principle and architecture's design of Long-Short Term Memory networks (LSTMs) to a *recurrent neural network (RNN)* called *LSTM-RNN*, training the model on the KDDCUP99 [64] dataset. After the evaluation of their system's performance, the resulting accuracy measures showed that this technique outperforms other existing state of the art techniques, as shown in Table 2.24:

Table 2.24: Experimental Results Achieved in [43]

Approach	Detection Rate	False Positives Rates	Accuracy
GRNN	59.12%	12.46%	87.54%
PNN	96.33%	3.34%	96.66%
RBNN	69.83%	6.95%	93.05%
KNN	45.74%	46.49%	90.74%
SVM	87.65%	6.12%	90.4%
Bayesian	77.6%	17.57%	88.46%
LSTM-RNN [43]	98.88%	10.04%	96.93%

In [41], the authors proposed a hybrid approach using both *ensemble methods* and *Long-Short Term Memory (LSTMs)* networks which was able to detect anomalies on system calls. It is able to merge multiple thresholding classifiers into a single one and has the advantage of learning the semantics and interactions of each system call compared with other techniques. The validity and performance of their approach was evaluated on various publicly available benchmark datasets. The results of this evaluation are shown in the following Table 2.25:

Table 2.25: Experimental Results Achieved in [41]

Benchmark Dataset	Area Under the Curve (AUC)	False Positives Rate	Detection Rate
KDDCUP99 [64]	99.4%	2.3%	100%
UNM	96.9%	5.5%	99.8%

In [78], a study was conducted by the authors in order to evaluate the performance of *Long-Short Term Memory recurrent neural networks (LSTM-RNN)* in detecting network anomalous traffic. The results of their evaluation which was conducted on KDDCUP99 [64] showed that this technique is able to learn and detect all the types of attacks present in the benchmark dataset.

In [86], a deep learning approach for intrusion detection using recurrent neural networks called *RNN-IDS* was introduced. Moreover, and in order to tune the performance of the approach, different number of neurons and learning parameters were tested. After comparison with other state of the art techniques like J48, artificial neural networks, random forests, support vector machines (SVMs) and many others, it was concluded that their technique outperforms them in terms of accuracy measures; with 99.53% accuracy using 80 hidden nodes (neurons) and 0.5 learning rate value on KDDTrain, 81.29% accuracy using the same configuration on KDDTest, and 64.67% of accuracy using the same configuration on KDDTest-21, achieving 11444 seconds of training time.

In [19], a novel approach was proposed in order to detect collective network anomalies on time-series using *Long-Short Term Memory Recurrent Neural Networks (LSTM-RNN)*. Their approach works differently in a way that it is trained on normal time-series data before predicting future time-series. Then, the prediction errors from the number of latest time-series steps is considered as collective anomaly if it is above a threshold. The model was trained on a time-series representation of the KDDCUP99 [64] benchmark dataset and the evaluation of this technique proves that it can detect collective anomalies efficiently. The evaluation results are shown in Table 2.26:

Table 2.26: Experimental Results Achieved in [19]

Threshold Value	True Positives Rate	Number of False Alarms
0.69	86%	0
0.66	94%	2
0.62	98%	16
0.52	100%	63

In [30], a novel network time-series anomaly detection technique was proposed by the authors using *stacked Long-Short Term Memory networks based on softmax activation function* classifier which is able to predict new packages' signatures. The evaluation of the approach was conducted on a real dataset created from a gas pipeline SCADA system, and the results showed that this technique outperforms other state of the art methods (Table 2.27).

Table 2.27: Experimental Results Achieved in [30]

Approach	Precision	Recall	Accuracy	F1_Score
Proposed Approach in [30]	94%	78%	92%	85%
BF	97%	59%	87%	73%
BN	97%	59%	87%	73%
SVDD	95%	21%	76%	34%
IF	51%	13%	70%	20%
GMM	79%	44%	45%	59%
PCA-SVD	65%	28%	17%	27%

In [42], and in order to build an IDS classifier using deep learning paradigm, a novel *Long-Short Term Memory Recurrent Neural Network (LSTM-RNN)* model using *Nadam Optimizer* was proposed. Several iterations of training were conducted where six other optimizers where tested and the latter one (Nadam) was chosen as the best one. After extensive evaluation and experimentation on KDDCUP99 [64] benchmark dataset, the following classification performance metrics shown in Table 2.28 were computed:

Table 2.28: Experimental Results Achieved in [42]

Optimizer	Accuracy	Recall	False Positives Rate	Precision	Efficiency
RMSprop	94.96%	97.70%	15.36%	95.98%	7.07
Adagrad	96.78%	99.14%	12.27%	96.88%	8.57
Adadelta	94.40%	99.07%	23.12%	94.22%	5.40
Adam	95.91%	98.72%	12.87%	96.75%	8.20
Adamax	97.37%	98.62%	10.64%	97.59%	9.30
Nadam	97.54%	98.95%	9.98%	97.69%	9.98

In [23], an improved network anomaly detection technique was proposed using a combination of *stacked convolutional neural networks (CNNs)* and *gated recurrent units (GRUs)* was proposed by the authors in which they detect anomalous system calls which is called *CNN-GRU*. After several experiments, they concluded that by stacking multiple CNN layers before using GRU layer improves the accuracy and execution time compared with approaches using only LSTMs. The results of their experimentation and evaluation are depicted in the following Table 2.29:

Table 2.29: Experimental Results Achieved in [23]

Approach	RNN Units	Training Time (Seconds)	Testing Time (Seconds)	Area Under the Curve (AUC)
GRU	200	376	444	66%
LSTM	200	4444	541	74%
CNN+GRU	200	390	441	80%
CNN+GRU	500	402	493	79%
CNN+GRU	600	413	533	81%

In [61], a novel network anomaly detection technique was introduced using *sequential autoencoder* alongside *Long-Short Term Memory (LSTM)* networks. First, autoencoders are used in order to reduce the data dimensionality and extract the necessary and important features. Then, LSTMs are used to classify these sequential data representations obtained from the autoencoder; using a fixed threshold which is based on a cross-validation process, resulting in the incoming network traffic being classified as either anomalous or normal.Moreover, their techniques is efficient in detecting both known and unknown (zero-day) anomalies and network attacks. Finally, through an extensive experimentation and evaluation, the results showed that their proposed approach has good performance, dynamicity, robustness, and scalability. Table 2.30 shows the performance metrics computed during this evaluation:

Table 2.30: Experimental Results Achieved in [61]

Approach	Threshold	F1_Score	Area Under the Curve (AUC)
LSTM	0.008	84.09%	95.19%
GRU	0.006	81.02%	94.78%
Bi-LSTM	0.008	84.61%	95.12%
NN1 - 1 Layer	0.008	83.52%	94.99%
LSTM AutoEncoder with Last Pooling	0.076	84.09%	95.07%
LSTM AutoEncoder with Max Pooling	0.076	85.38%	95.12%
LSTM AutoEncoder with Mean Pooling	0.079	80.72%	94.71%
[61]	0.076	85.38%	95.12%

In [25], in order to detect network anomalies and attacks in *Internet of Things (IoT) fog-to-things communications*, a novel approach was introduced using *Long-Short Term Memory (LSTMs)*. This proposed technique is able to efficiently detect critical attacks and threats which target IoT devices, and more precisely, the ones which exploit vulnerabilities found on wireless communication devices. The experimentation of their work on two different scenarios prove the performance and efficiency of their technique compared to other state of the art techniques. Table 2.31 shows the results of this extensive evaluation:

Table 2.31: Experimental Results Achieved in [25]

Approach	Benchmark Dataset	Accuracy	Recall	Precision
LSTM [25]	UNBCIC [7]	99.91%	99.96%	99.85%
LSTM [25]	AWID	98.22%	98.9%	98.5%
LR	UNBCIC [7]	90%	99%	89.11%
LR	AWID	84.87%	90%	85%

In [80], a novel intrusion detection system for *software defined networking (SDN)* using *gated recurrent unit recurrent neural network (GRU-RNN)* was introduced. The proposed approach was tested and evaluated using the NSL-KDD benchmark dataset and the results of this evaluation showed that this proposed technique achieves higher accuracy compared to other state of the art techniques, using only a set of six features. Table 2.32 shows these results:

Table 2.32: Experimental Results Achieved in [80]

Approach	Accuracy
VanillaRNN	44.39%
SVM	65.67%
DNN	75.9%
GRU-RNN [80]	89%

– **AutoEncoders:** As shown in Figure 2.22, there are four main works and publications that use mainly autoencoders in order to detect network anomalies.

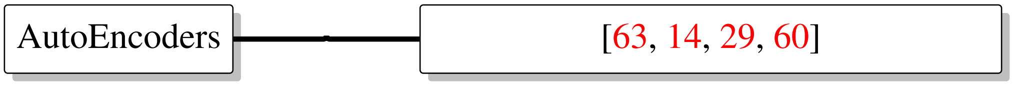


Figure 2.22: Publications Using AutoEncoders for Their Anomaly Detection Approaches. [10, 45]

In [63], a novel network anomaly detection techniques was proposed by the authors which combined both *autoencoders* and *density estimation*. First, an autoencoder reproduced the input data in the output layer, resulting into a compressed representation of the data within the smallest hidden layer. Then, they considered anomalies the ones resulting with lower density within the hidden layer by studying two scenarios for the density function: *Guassian* and *full kernel density estimation*. Their approach was evaluated on the benchmark NSL-KDD dataset and shows that the proposed approach outperforms previous best results. Table 2.33 shows the results of this evaluation.

Table 2.33: Experimental Results Achieved in [63]

Attack Type	Approach	Area Under the Curve (AUC)
DoS	OCAE	96%
DoS	OCCEN [63]	95.6%
DoS	OCKDE [63]	97.4%
R2L	OCAE	90.9%
R2L	OCCEN [63]	83.9%
R2L	OCKDE [63]	89.1%
U2R	OCAE	92.8%
U2R	OCCEN [63]	88.8%
U2R	OCKDE [63]	94.5%
Probing	OCAE	97.1%
Probing	OCCEN [63]	98.6%
Probing	OCKDE [63]	98.7%

In [14], by using the *reconstruction probability* from *variational autoencoder*, a novel anomaly detection technique was proposed called *variational autoencoder (VAE)*. This probabilistic value measures the variation of the variables' distributions. In contrary to *reconstruction error* which is generally used by autoencoders, *reconstruction probability* is considered by the authors of this work as more suitable to be used as an anomaly score metric. The results of their evaluation proved that their proposed technique outperforms the classical approaches using either autoencoders or principal component analysis (PCA). This evaluation was conducted on KDDCUP99 [64] benchmark dataset and the results are depicted in the following tables; Table 2.34 shows the results obtained on traffic labelled as benign, whereas Table 2.35 shows the results obtained on traffic labelled as malicious:

In [29], a novel anomaly detection technique using *deep autoencoder (DAE)* was introduced called *DAE-IDS*. Their model training was conducted in a *greedy layer-wise* fashion in order to avoid overfitting and local optima. The evaluation was conducted on KDDCUP99 [64] benchmark dataset and the results showed that their proposed approach outperforms other state of the art deep learning existing techniques in terms of accuracy, detection rate, and

Table 2.34: Experimental Results Achieved in [14] on normal traffic.

Attack Type	AUC ROC	F1_Score
DoS	79.5%	98.1%
R2L	77.7%	40.6%
U2R	78.2%	32.4%
Probing	94.4%	79.1%

Table 2.35: Experimental Results Achieved in [14] on malicious traffic.

Attack Type	AUC ROC	F1_Score
DoS	74.4%	97.9%
R2L	78.6%	38.9%
U2R	92.1%	34.7%
Probing	97%	72%

false positives rates, as shown in Table 2.36:

Table 2.36: Experimental Results Achieved in [29].

Method	True Positives Rates	Accuracy
DAE-IDS [29]	94.42%	94.71%
DBN	92.33%	93.49%
AutoEncoder+DBN	92.20%	92.10%

In [60], a hybrid anomaly detection technique was proposed which combined both *stacked autoencoders (SAE)* and *support vector machines (SVM)* called *DL-SVM*. In more details, the authors used *stacked autoencoders (SAE)* for the dimensionality reduction of all the features and extract only those of interest. whilst *support vector machines (SVM)* are used as a binary classifier in order to classify the data into either anomalous or benign network traffic. The experimentation and evaluation was conducted on UNBCIC [7] benchmark dataset and the results showed that their approach outperforms state of the art existing techniques were only SVMs were used. The following Table 2.37 shows these results:

b) **Self-taught Learning:** in this category, there is one main work and publication that is of interest as shown in Figure 2.23.

Table 2.37: Experimental Results Achieved in [60].

Method	Accuracy	Precision	Recall	False Positives Rate	ROC	F1_Score
PCA-SVM	85.6%	88%	84.9%	13.6%	85.6%	84.7%
DL-SVM [60]	90.2%	90.3%	9.8%	90.2%	90.3%	80.6%

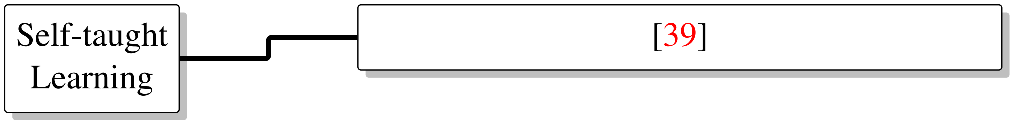


Figure 2.23: Publications Using Self-taught Learning for Their Anomaly Detection Approaches. [10, 45]

In [39], a novel flexible and efficient network anomaly detection technique which uses *self-taught learning (STL)* was introduced by the authors. The evaluation of this work was conducted on the *NSL-KDD* benchmark dataset and was compared with other state of the art anomaly (*softmax regression (SMR)*) detection techniques in terms of accuracy, precision, recall, and f-measure. The following Table 2.38 shows the results of this evaluation:

Table 2.38: Experimental Results Achieved in [39].

Method	Accuracy	Precision	Recall	F1_Score
SMR (2-class)	78%	97.3%	96.2%	96.8%
SMR (5-class)	75%	86%	62%	73%
STL [39] (2-class)	98%	99%	97%	98%
STL [39] (5-class)	98%	84%	70%	75%

2.2.1.9 Clustering-based Anomaly Detection Techniques

Clustering based anomaly detection techniques on the other hand are categorized according to our litterature review and to these two surveys into three sub-categories:

a) K-Means: where a parameter *k* which represents the number of clusters to be detected is defined generally by the user and aims to group/cluster the data into these groups/clusters.

expectation-maximization clustering (EM): is kind of an improvement and extension to k-means clustering where an object is assigned to a cluster which it is similar to based on this cluster's mean.

outlier detection: is a clustering technique that detects patterns of data points that do not conform to the expected behaviors (outliers). As shown in Figure 2.24.

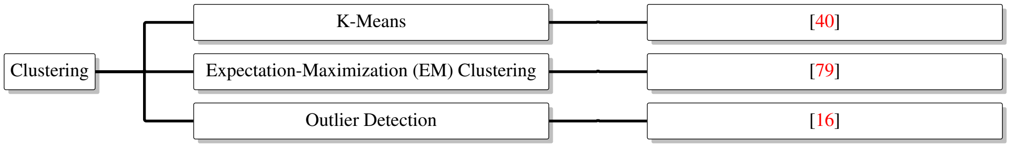


Figure 2.24: Publications Using Clustering for Their Anomaly Detection Approaches. [10, 45]

a) ***K-Means Clustering:*** In [40], a novel hybrid anomaly detection approach was proposed in order to detect network anomalies in *content-centric networks (CCNs)*. This approach combined both particle swarm optimization (pso) and k-means clustering during the training phase in order to choose the best number of clusters k. During the second phase, it used fuzzy logic alongside two distance metrics in order to detect anomalies. The experimental results proved that this combination during the training phase achieves better accuracy results when the optimal number of clusers k is chosen, and it also outperforms other state of the art methods.

b) ***Expectation-Maximization (EM) Clustering:*** In [79], a comparative study in terms of accuracy and false positives rates has been conducted between *k-means*, *k-meloids*, *EM clustering*, and *distance-based outlier detection* in order to decide which one is best fit for intrusion detection on network data. The experimental results showed that both *distance-based outlier detection* and *EM clustering* approaches outperform the other techniques as shown in Table 2.39:

Table 2.39: Experimental Results Achieved in [79].

Method	**Accuracy**	**False Positives Rate**
k-Means	57.81%	22.95%
Improved k-Means	65.4%	21.52%
k-Meloids	76.71%	21.83%
EM Clustering	78.06%	20.74%
Distance-based Outlier Detection	80.15%	21.14%

c) ***Outlier Detection:*** In [16], an outlier detection technique was proposed in order to detect anomalies on network traffic datasets. The technique started by selecting the appropriate non-redundant features by applying generalized entropy metric. Then, a tree-based clustering technique is applied to generate a set of reference points. Finally, it computed outlier scores using outlier rank function to detect possible outliers. The experiments were conducted on both KDDCUP99 [64] and NSL-KDD datasets in order to evaluate their approach, and the results show that their proposed technique outperforms other state of the art techniques.

2.2.2 Statistical Anomaly Detection

In this category of network anomaly detection techniques, there are two major works that we deem as important for our study, as shown in Figure 2.25

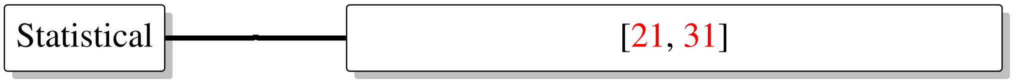


Figure 2.25: Classification of Statistical Anomaly Detection Approaches. [10, 45]

In [21], a statistical network anomaly detection technique using *multivariate statistical process controll (MSPC)* technique based on *principal component analysis (PCA)* was thoroughly studied and evlauted in order to assess its accuracy metrics.They also concluded at the end of their study that MSPC-PCA outperforms the standard MSPC approach and other state of the art techniques.

In [31], an anomaly detection technique that combines both *principal component analysis (PCA)* statistical and *ant colony optimization* metaheuristic techniques was presented by the authors. These two combined techniques generate network traffic profile or signature called *Digital Signature of Network Segment using Netflow analysis (DSNSF)* and is considered by the authors as ***normal traffic signature***. Then, this generated signature was compared with real-world network traffic by introducing a modified version of the *Dynamic Time Warping* metric in order to differentiate between normal and abnormal traffic. The experiments were conducted on real network traffic in order to evaluate the approach and the results showed that this technique improves the detection rate by maintaining a good and low false positives rate.

2.2.3 Soft-Computing Anomaly Detection

There are many works that have been achieved using soft-computing paradigm in order to build network anomaly detection tools, but in this document and related to our work, we consider only one major work as prominent as shown in Figure 2.26:

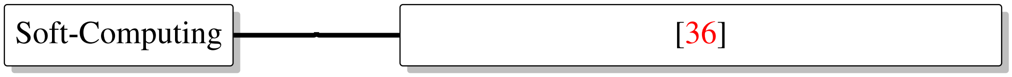


Figure 2.26: Classification of Soft-Computing Anomaly Detection Approaches. [10, 45]

In [36], a network anomaly detection approach that combined both a *Genetic Algorithm* and *Fuzzy Logic* is introduced. First, the technique defined a batch interval of five minutes; where four nominal features (src/dst IPs and Ports) are converted into numerical single values using *Shannon Entropy* metric, and numerical features (src/dst number of bytes and packets) were summerized into single values using their sum during this interval; the results of this step were considered by the authors as the normal traffic behavior/profile signature. Then, *Genetic Algorithm* was used to generate the network's predicted/expected behavior/profile signature called *Digital Signature of Network Segment using Netflow analysis (DSNSF)* using the same set fo features. After that, an adaptive threshold was defined on the normal traffic profile by using the *exponential weighted moving average (EWMA)* metric. After that, fuzzy logic was used with *Guassian Membership Function* in order to compute anomaly scores on each of these features using the pre-computed profiles (normal, predicted/expected, and thresholds). Finally, the same metric used for computing the thresholds (EWMA) was used on these computed anomaly scores in order to decide whether one of these features was anomalous or normal at a specific time. The results obtained by applying this technique on real network traffic flows achieved an accuracy of 96.53% and a false positives rate of 0.56%, the results also proved that this approach outperforms other state of the art techniques.

As a conclusion to this litterature review on the most prominent existing network anomaly detection techniques we can say that each technique has its own advantages and limitations. Machine learning techniques have the advantage of better accuracy results and attack detection rates over the other techniques. However these techniques suffer from scalability issues since they often require training iterations (supervised) and are time consuming (unsupervised), delaying the delivrance of our analytics results to our partners. Moreover, supervised techniques also often require to know the attack signature or pattern in order to detect it which makes them weak and sometimes unable to detect zero-day threats. Statistical and sof-computing techniques on the other hand are better to use when it comes to scalability, since they are fast compared to machine learning techniques (statistical techniques) and they are much better to use in an environment with plenty of noise and uncertainties (soft-computing techniques).

Chapter 3

Anomaly Detection

In this chapter we present our porposed network anomaly detection approach in more details. We first give an overview of the approach, then we explain each step in full details and present the predictive algorithm that we proposed.

3.1 Approach Overview

An overview of our approach is represented in Figure 3.1, which consists of four major steps: *feature extraction*, *adaptive thresholding*, *predictive analysis*, and *anomaly detection*. In the *feature extraction* step, the features that are considered as paramount to our anomaly detection system are extracted. More specifically, we extract a set of eight features that can be found in any IDS logs in order to achieve system *interoperability*. We further summarize the network behavior/profile for each of the chosen features within a time batch of five minutes. The obtained eight time series representations are considered as the first input and observed network behavior/profile to our anomaly detection system.

In the next phase, an adaptive threshold using the *exponential weighted moving average (EWMA)* [56] statistical metric is calculated for each of the pre-computed network behavior profiling time-series. The reason of using such technique is that daily users' Internet connections are in some way *auto-correlated* over time [56]; meaning that each user connection on the Internet is mostly affected by his/her past interactions and not completely random, and the interactions' influences (weights) on the current connection decreases back in time. Therefore, we utilize a metric that best captures and defines these weights.

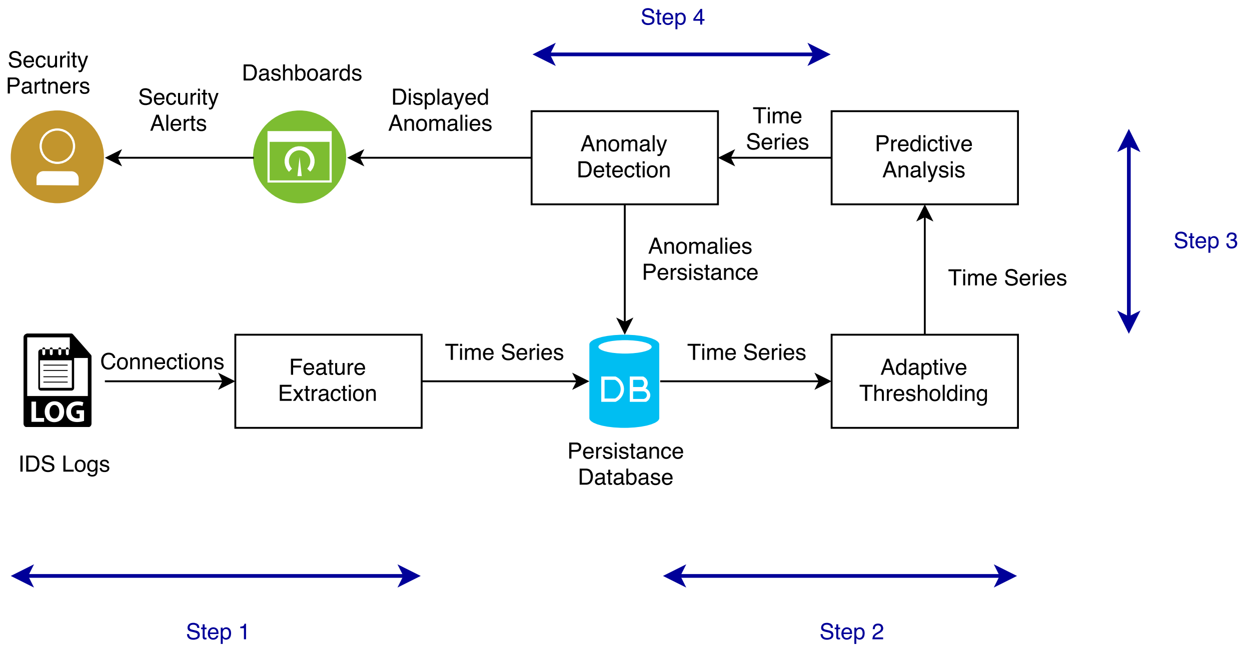


Figure 3.1: Proposed Anomaly Detection Approach

In the *predictive analysis* step, we consider a time-window of one hour and try to predict the expected behavior of each of these network features within the same time frame without any noise or seasonal trends. Consequently, we utilize the *particle swarm optimization (PSO)* optimization algorithm, which is designed to mimic the social behavior of some animals (e.g., birds and fishes) [27]. We modify and adapt this algorithm to our objectives by defining the *fitness* function as the *augmented Dickey-Fuller (adfuller)* test [62], which examines the stationarity of a given time series as input.

Finally, in the last step, we take as input the three pre-computed time series for each feature and compute the corresponding anomaly scores. More specifically, a *fuzzy logic* inference system which has two main *Fuzzification* and *Defuzzification* phases is utilized. In *Fuzzification phase*, a membership function (Gaussian membership function) is applied on the time series, in which the anomaly scores are computed. Afterwards, in the *Defuzzification phase*, the adaptive thresholding is applied on these scores, and further according to obtained scores and their representative threshold, the associated features are flagged as either anomalousor non-anomalous.

3.2 Anomaly Detection by Time-series

In this section, we describe our time-series based anomaly detection approach, which is combined with an optimization algorithm (particle swarm optimization). Moreover, we describe the dataset used for our experimentation and demonstrate how to validate the performance of our approach.

3.2.1 Feature Extraction

Prior anomaly detection process, a set of features should be extracted. This task is primordial as it affects the final results of the approach tremendously. Many works have been achieved in the field of network anomaly detection in order to determine what are the features that give better insights of the network behavior [34, 46, 36]. In this work, we aim at not merely gather a set of attributes that best describe the network profile and behavior, but also obtain an anomaly detection system, which works with any IDS logs, such as BRO, Snort and Surricata IDSs. Achieving the latter property provides the *interoperability* to our proposed system. Therefore, we target the possible features that are present in any IDS system. According to the aforesaid criteria and alongside with some literature review [46] regarding this objective, we extract the set of eight features of *Originator IP Address*, *Responder IP Address*, *Originator TCP/UDP Port*, *Responder TCP/UDP Port*, *number of bytes sent by the originator IP*, *number of bytes sent by the responder IP*, *number of packets sent by the originator IP*, and *number of packets sent by the originator IP*.

On the other hand, we receive the connection logs in the order of millions (sometimes hundreds of millions connection logs) per day, and therefore there would be a scalability issue to process all of these connection logs in a reasonable time. Therefore, to address the scalability issue we propose to use time-batches splitting [36] of five minutes to reduce the amount of processing time.

Our approach is a time-series based technique, which accepts numerical values in order to build the time-series representations. However, the type of the first four selected features (*Originator and Responder IP Addresses*, and *Originator and Responder TCP/UDP Ports*)is nominal. In order to convert nominal attributes to numerical representations, inspired by [46], we choose to use *Shannon Entropy* [36, 46, 72] as a summarization tool to represent the distribution of the features. Additionally, the authors prove that feature distributions enable the detection of a wide range of network anomalies, including unknown anomalies. The authors further demonstrate that studying these distributions can lead to an automatic unsupervised classification (clustering) represented in Table 3.1.

Table 3.1: Effect of various anomalies on different feature distributions [46]

Anomaly Type	**Description**	**Affected Distributions on**			
		Src IP	**Dest IP**	**Src Port**	**Dest Port**
Denial of Service (DoS)	A single infected host is used to attack a victim and disable some services	Low	Low	-	-
Distributed Denial of Service (DDoS)	Multiple infected hosts and internet connections are used to target a single host	High	-	-	-
Port Scanning	A large amount of traffic is sent to a small amount of destination IPs using large amount of destination ports	-	Low	-	High
Network Scanning	A large amount of traffic is sent to a large amount of destination IPs using a small amount of destination ports	-	High	-	Low
Outage Events	The traffic amount goes down due to the fact that a service went down, equipment failure, or maintenance operations.	≈ 0	≈ 0	≈ 0	≈ 0
Flash Crowd Attacks	A huge surge in the amount of traffic going to one single destination IP		Low		Low

The intuition behind studying traffic features distributions is that the majority of network attacks affect the distributions of aforementioned attributes. For instance, consider a DDOS attack in which multiple source hosts (compromised machines) flood the targeted destination machine until the aimed goal is achieved (take down a specific service). By looking at the distributions of both source and destination IPs it could be inferred whether there is any spike/surge in the source IPs distribution and a drop-down in the destination IPs distributions, which leads to the detection of anomalies/attacks.

Shannon Entropy is a good metric to represent the degree of dispersal or concentration of any distribution and quantify these changes in a single numerical value. Given an attribute $X = \{x_1, x_2, \ldots, x_n\}$, where x_i is the frequency of the i_{th} attribute sample with the targeted time-batch (five minutes), the *Shannon Entropy*, $H(X)$, is computed as follows [36, 72]:

$$H(X) = \sum_{i=1}^{n} \left(\frac{x_i}{\sum\limits_{i=1}^{n} x_i} \right) . \log_2 \left(\frac{x_i}{\sum\limits_{i=1}^{n} x_i} \right) \tag{3.1}$$

Finally, we also need to quantify the other four numerical values (sent bytes and packets from both originator and responder IPs) into a single value. For this purpose, inspired by [36] we simply consider the cumulative sum of each one of their values during the specific time-batch. By performing all the pre-processing steps, we end up with a full time-series representations (eight time-series, one for each feature) for the observed network behavior over the time.

3.2.2 Adaptive Thresholding

The next step of our approach consists of finding a metric to compute the thresholds of our pre-computed time-series representations. We deal with Internet connections, which are auto-correlated; meaning that the behaviors of users using a specific network are not random and thus they are affected mostly by their past interactions on the Internet. Therefore, we use the *exponential weighted moving average (EWMA)* as such metric. *EWMA* is a statistical metric that allows analysts to monitor the average of the data by giving more importance to the most recent data observations. This weighting system is computed in exponential fashion using all prior data observations. Considering that we have a time-series based representation of data observations in a specific time-window, *EWMA* is computed as follows [56]:

$$EWMA_t = \begin{cases} X_1, & \text{, if: } t = 1 \\ \alpha . X_t + (1 - \alpha) . X_{t-1} & \text{, if: } t > 1 \end{cases} \tag{3.2}$$

where X_1 is the network observation at the time $t = 1$, X_t is the current network feature observation, X_{t-1} is the most recent (previous) network feature observation, and α is the weighting exponential factor which falls between 0 and 1 and is computed as $\alpha = e^{-\left(\frac{\epsilon}{\sigma}\right)}$ [6], where ϵ represents the time elapsed since the time-window started until the current observation, and σ represents the size of the time window (in our case it is set to one hour).

In addition, *EWMA* is a good statistical metric for measuring *historical volatility*, which represents the degree of dispersion for a given dataset over a defined period of time [8]. There are two main approaches to measure the volatility of data over time, namely *Implicit* and *Historical* [8]. *Historical* approaches assume that past data observations are essential for the prediction of future observations. *Implicit* approaches, on the other hand, ignore the past observations completely, and try to predict future observations based on the current observations. The fact that we are dealing with Internet connection logs implies that there are some auto-correlations (inter-connections) between the current and the past (most recent) internet connections of the users.

These summarization techniques have a major drawback in the way that all past observations during the time-window will have the same influence or weighting system [8]. This weakness would significantly influence and affect our results. Let's take an example where we have Internet connection logs over one month, and during let's say the 24^{th} day of this month a DDOS attack occurred affecting all the source IPs in the logs that we receive. The next day (25^{th}), this DDOS attack led the services to shut down or outage of most services on the targeted hosts (source) or IPs. If one applies the approach where the same weights are applied to all the observations of that month, this will terribly affect our system in the way that when computing the measure for the 25^{th} day, and considering that the time window is of one month, the same importance will be given to the day when the attack occurred (previous day) but also to all previous days of the month. However by applying *EWMA* and computing the historical volatility measure for the day after the attack occurred (25^{th}), the previous and most recent day (24^{th}) will be given much more weight and importance over the previous days of the month, leading to capturing more patterns of the attack from the network flow attributes.

3.2.3 Predictive Analysis

The next step of our anomaly detection system consists of predicting the normal network profile or behavior. One common and simple approach that is generally applied would be computing the differences between observed (real) network traffic and computed (fixed) thresholds and in the case of positive results, infer that the real traffic is anomalous (above thresholds), otherwise no anomaly is detected. However, since we are dealing with daily Internet connection logs, which are not pre-filtered (or pre-cleaned); meaning that they contain a huge number of false positives. For instance, each day thousands or even hundreds of thousands of users and academics connect to Google website, if we consider only the two pre-computed steps (feature extraction and adaptive thresholding), we would end up with a lot of misleading attacks classifications and false positives (it could be classified as a DDOS attack). Therefore, we extend these two steps with another predictive step, which tries to delete most of these false positives; in other words, given a real network raw traffic, we ask our system *how much clean the network traffic should look like?*

In order to do so, we opt for using an optimization algorithm, namely particle swarm optimization (PSO) [27, 58, 84] as shown in Algorithm 1, which achieves the desired goal of this step. PSO is a meta-heuristic global optimization technique that belongs to the family of *swarm intelligence* algorithms [27]. It is based in analogy with the social behavior of certain animals and most precisely *bird flocks* and *fish schools*. In PSO, the ensemble or set of possible solutions to the optimization task are called a *swarm* and each element of this swarm is called a *particle*. These particles move and change their positions in the search (parameter) space based on their own and neighbors' best performances. Therefore, this evolution process of the swarm is based on two main principles: *cooperation* and *competition* among these particles across multiple generations (several iterations).

Algorithm 1 Particle Swarm Optimization Algorithmic Description [27, 58, 84]

```
• Initialization Process: For each of the N particles:
Initilize the position $x_i(0) \forall i \in N$
Set the particle's best personal or local position as its initial position: $p_i(0) = x_i(0)$
Compute the fitness of the particle and if $f\left(x_j(0)\right) \geq f\left(x_i(0)\right) \forall i \neq j$ initialize the swarm global best as: $g = x_j(0)$
• Repeat the following steps until the pre-defined criterias are met:
Update the particle velocity according to the equation (3.6).
Update the particle position according to the equation (3.5).
Compute the fitness of the particle $f\left(x_i(t+1)\right)$
if $f\left(x_i(t+1)\right) \geq f\left(p_i\right)$, update the particle's personal or local best as: $p_i = x_i(t+1)$
if $f\left(x_i(t+1)\right) \geq f\left(g\right)$, update the swarm's global best as: $g = x_i(t+1)$
• Once the stopping criterias are met and the iterations are stopped, the best solution to the problem is represented by g.
```

Accordingly, in PSO, each particle is defined in the D-dimensional search space, where D represents the set of parameters to be optimized. The position of the i_{th} particle is defined by the following vector $x_i = [x_{i1}, x_{i2}, x_{i3}, \ldots, x_{iD}]$, and the population (swarm) of N candidate solutions is $X = \{x_1, x_2, x_3, \ldots, x_N\}$. In their journey of finding the optimal solution, each particle updates its own position using $x_i(t+1) = x_i(t) + v_i(t+1)$, where *t* and *t+1* represent two successive iterations of the algorithm and v_i represents a vector called the *velocity*, which governs the way the particle changes its position across the search space. The velocity parameter is defined according to three factors: (i) *Inertia or Momentum:* represents the previous velocity of the same partcile (previous iteration of the algorithm) and helps prevent from drastic position changes. (ii) *Cognitive Component:* represents the possibility of the partcile to return to the previous position. (iii) *Social Component:* identifies the ability of the particle to move forward the best solution of the whole swarm.

Based on these definitions, the velocity of the i_{th} particle is defined as $v_i(t+1) = v_i(t) + c_1\,(p_i - x_i(t))\,R_1 + c_2\,(g - x_i(t))\,R_2$, where p_i represents the particle's best solution (local or personal best), while g represents the global best (the overall best solution found by the whole swarm). The two real-values of c_1 and c_2 are called *acceleration constants* that define the way by which the particle moves toward the global best solution. On the other hand, R_1 and R_2 represent respectively two diagonal matrices of randomly generated numbers from a uniform distribution in the interval $[0, 1]$. For both *cognitive* and *social* components to influence the particle's velocity in a stochastic way, the acceleration constants are generally set to 2 so that they meet the $0 \leq c_1.c_2 \leq 4$ criteria [58].

In order to get the PSO work correctly, the *fitness* or *objective* function, which defines the stopping criteria of the optimization task needs to be defined. In this work, we aim our predicted time-series to achieve stationary. The reason behind this logic is the fact that time-series statistical summarization techniques (such as the mean, variance, and standard deviation) do not give consistent results due to the presence of trends and seasonal effects (presence of periodic fluctuations). While time-series are stationary, they are not bounded by the time and summary statistics are more consistent. In addition, they can easily be modelled by statistical modelling approaches (e.g., forecasting techniques). Thus, it is always recommended to check if a time-series representation is stationary, and if not, make it stationary by the removal of any trends and seasonal effects before moving to analysing the *residual effects* [5].

One of the most frequently used stationary tests techniques is the *Augmented Dickey-Fuller Test (adfuller)* [62], which is a type of statistical test and autoregressive model for stationary that belongs to the *unit root* tests family. In *adfuller* algorithm, a time-series is considered as non-stationary if one of its monomials is equal to 1; this is equivalent to say that it can be defined in function of some specific trend (case of null hypothesis satisfied). On the other hand, if this test is rejected (alternate hypothesis satisfied), none of its monomials would be equal to 1, therefore it is considered as stationary [5].

In our work, once we build our representation of the network traffic behavior or profile (feature extraction phase), we then call *adfuller* test on each of the features' time-series. If at least one of the eight time-series is found as non-stationary (null hypothesis satisfied), then we trigger our PSO implementation in order to remove the presence of any trends and seasonal effects on that specific time-series. Otherwise (in the case of alternate hypothesis satisfied and thus the null hypothesis is rejected), if all the time-series are stationary, we do not trigger our PSO implementation and consider for our anomaly detection module only the observed behavior and the computed adaptive thresholds. Algorithm 2 best describes this statement.

3.2.4 Anomaly Detection

The final step of our anomaly detection system consists of using all the pre-computed time-series for detecting anomalies on each feature. For that purpose we employ *Fuzzy-Logic* [36, 59] or fuzzy inference system for computing anomaly scores for each feature. In the fuzzification step, we compute anomaly scores ranging between 0 and 1 on the pre-computed time-series (observed, thresholds, and predicted behaviors) using the *Gaussian Membership Function* as defined in equation 3.7. If the prediction step was not triggered, we compute these scores using the equation 3.8. After the fuzzification process is terminated, we move on to the defuzzification process, where we define adaptive thresholds on these anomaly scores using the same technique presented for adaptive thresholding step (*EWMA*), and then flag each feature as anomalous (flag it as 1 if the anomaly score is above the threshold) or not (flag it as 0 if the anomaly score is below the threshold). The Gaussian membership function is defined as follows [36, 59]:

$$score_k = 1 - e^{\frac{-(x_k - y_k)^2}{2.\epsilon_k^2}} \tag{3.7}$$

where k ranges form 0 to 8 and represents the corresponding feature, x_k represents the observed or expected behavior, y_k represents the predicted feature behavior in the case our predictive step is triggered, and ϵ_k represents the computed adaptive threshold for that feature. However, if the prediction step is not triggered, the Gaussian membership function is defined as follows:

$$score_k = e^{-(x_k - \epsilon_k)^2} \tag{3.8}$$

Algorithm 3 describes how anomalies are detected on each feature. At the end of this process, we end up with each connection extended with eight anomaly scores and flags (one for each feature), as depicted in Table 3.2, where each connection is considered as anomalous if at least one of these eight flags is set to 1.

Algorithm 2 Proposed Predictive Algorithm

```
$InputVector \leftarrow ObservedSeries$
for each $FeatureTS \in InputVector$ do
  $InputData \leftarrow Feature - Time - Series$
  $Stop \leftarrow False$
  $ResultingTimeSeries \leftarrow InputData$
  $iCounter \leftarrow 0$
  while $Stop = False$ and $iCounter \leq 20$ do
    $p\_value \leftarrow \text{ADFULLER}(InputData)$
    if $p\_value > 0.5$ then
      $InputData \leftarrow \text{PSO}(InputData)$
    else
      $ResultingTimeSeries \leftarrow InputData$
      $Stop \leftarrow True$
    end if
  end while
end for
end
```

Algorithm 3 Anomaly Detection Fuzzy Inference Algorithm

```
$InputVector \leftarrow ObservedSeries$
for each $FeatureTS \in InputVector$ do
  for each $Obse, Threshl, Pred \in FeatureTS$ do
    $S \leftarrow$ GUASSIANFUN$(Obs, Threshl, Pred)$
    $TH \leftarrow$ ANOMALY-THRESHL$(S)$
    for each $score, threshold \in S, TH$ do
      if $score \geq threshold$ then
        $anomaly_{flag} \leftarrow 1$
      else
        $anomaly_{flag} \leftarrow 0$
      end if
    end for
  end for
end for
end
```

Table 3.2: Anomaly scores and flags

Anomaly	Features
Scores (between 0 and 1)	id_orig_h_anomaly_score, id_resp_h_anomaly_score, id_orig_p_anomaly_score, id_resp_p_anomaly_score, orig_bytes_anomaly_score, resp_bytes_anomaly_score, orig_pkts_anomaly_score, resp_pkts_anomaly_score
Flags (either 0 or 1)	id_orig_h_anomaly_flag, id_resp_h_anomaly_flag, id_orig_p_anomaly_flag ,id_resp_p_anomaly_flag, orig_bytes_anomaly_flag, resp_bytes_anomaly_flag, orig_pkts_anomaly_flag, resp_pkts_anomaly_flag

Chapter 4

Design, Implementation and Evaluation

In this chapter, we first describe the implementation of *Daedalus*, which is designed to extract cyber threat intelligence out from BRO NIDS stream of logs. Then, we evaluate our proposed algorithm and give insights of the proposed dashboards.

4.1 Design and Implementation

In this section, we present the implementation details of *Daedalus*. Figure 4.1 shows a technical overview of the proposed tool. *Daedalus* is built upon the following open source tools:

a) Apache Spark: in order to improve the computation.

b) MongoDB: to store only the results of our analytics.

c) Elasticsearch: also for the storage and fast consultation, searching, and indexing of the data.

d) Grafana: which is built upon Elasticsearch in order to create the needed visualisations and dashboards.

e) Docker: each of the previously cited tools (MongoDB, Elasticsearch, and Grafana) are deployed using dedicated docker containers.

f) Maxmind: used in order to geolocate public IP addresses.

Each of the open source tools mentioned above was chosen after a deep analysis of their importance to our work:

a) Apache Spark is a highly scalable computation engine that allows its users to distribute huge computation accross multiple computation nodes within a Spark Cluster. In this work, we only deploy it on a standalone server (processing server) but it can be deployed in the future as a cluster. We use Spark mainly to parse a stream of BRO NIDS connection logs (both csv and json formats); Apache Spark offers a streaming API, which enables us to achieve this task within very short delays.

b) MongoDB is used as a storage database in order to store our analytics' results only. We do not store raw logs as we receive them due to the fact that MongoDB is not scalable enough to save these tremendous logs. We also chose MongoDB because it is a JSON-based storage database, which makes it easy to query and comprehend. The raw connections are on the other hand stored as compressed files on our storage server.

c) Elasticsearch is used for mainly two reasons. The first one is to enable fast search and aggregation queries over the results of our analytics. The second reason is because it is necessary for creating visualisations for our front-end platform.

d) Grafana depends on Elasticsearch (as mentioned above) and allows us to create fancy visualisations (barcharts, piecharts, time-series visualisations, data tables, geo-maps, etc...) on top of our indexed data in Elasticsearch. In addition to that, it allows us to achieve many security objectives; it has a set of security functionalities that enable us to create security credentials and access restrictions for each of our security partners.

e) Docker containers are used to deploy each of the tools mentioned above on an isolated environment for two reasons. First, some of these services are already deployed on the dedicated processing server (like Elasticsearch for other projects); this can lead to a failure during the installation of our tool due to conflict with the dedicated ports, also if we deploy our data on the same existing Elastic node it will disclose data to our security partners. The second reason is related to security measures. Docker containers are completely isolated from the server's OS and even if it is damaged in some way, it does not affect the OS.

f) Maxmind Geolocation database is used to geolocate public (routable) IP addresses from the resulting BRO connections (both originator and responder IP addresses). We have bought a one year license from Maxmind that gives us monthly updates on both *City* and *ISP* databases. With both of these databases, we get in exchange all the information needed about each IP address fed to the database; mainly: Internet Service Provider (ISP), Organisation (Org), Autonomous System Number (ASN), Country (country name and code), City, lattitude, and longitude.

All the BRO NIDS connection logs received from our partners are first stored on our dedicated storage server, then, each week's data is transferred securely (using ssh) to our processing/computation server in order to analyze it. Once the analysis is done, the raw documents are deleted from the computation server in order to minimize the disk usage on the latter one. Our proposed tool is developed using Python programming language. First, we parse the connections received each week using Apache Spark streaming API deployed on top of Python programming language. From this step we get the needed set of features for our anomaly detection technique to work properly, all of the four steps of our technique are also implemented using Python Pandas API alongside Apache Spark computation tool; this will allow us to execute our proposed approach in a timely manner. In addition to its streaming API, Apache Spark also offers us the capability of integrating both MongoDB and Elasticsearch on top of it; this will enable us in addition to faster computation to store and access our analytics results fast and quickly on both these databases. Finally, for the geolocation part, we use a Python-based API (*geoip2*). The overall framework combining those components together is presented in Figure 4.1.

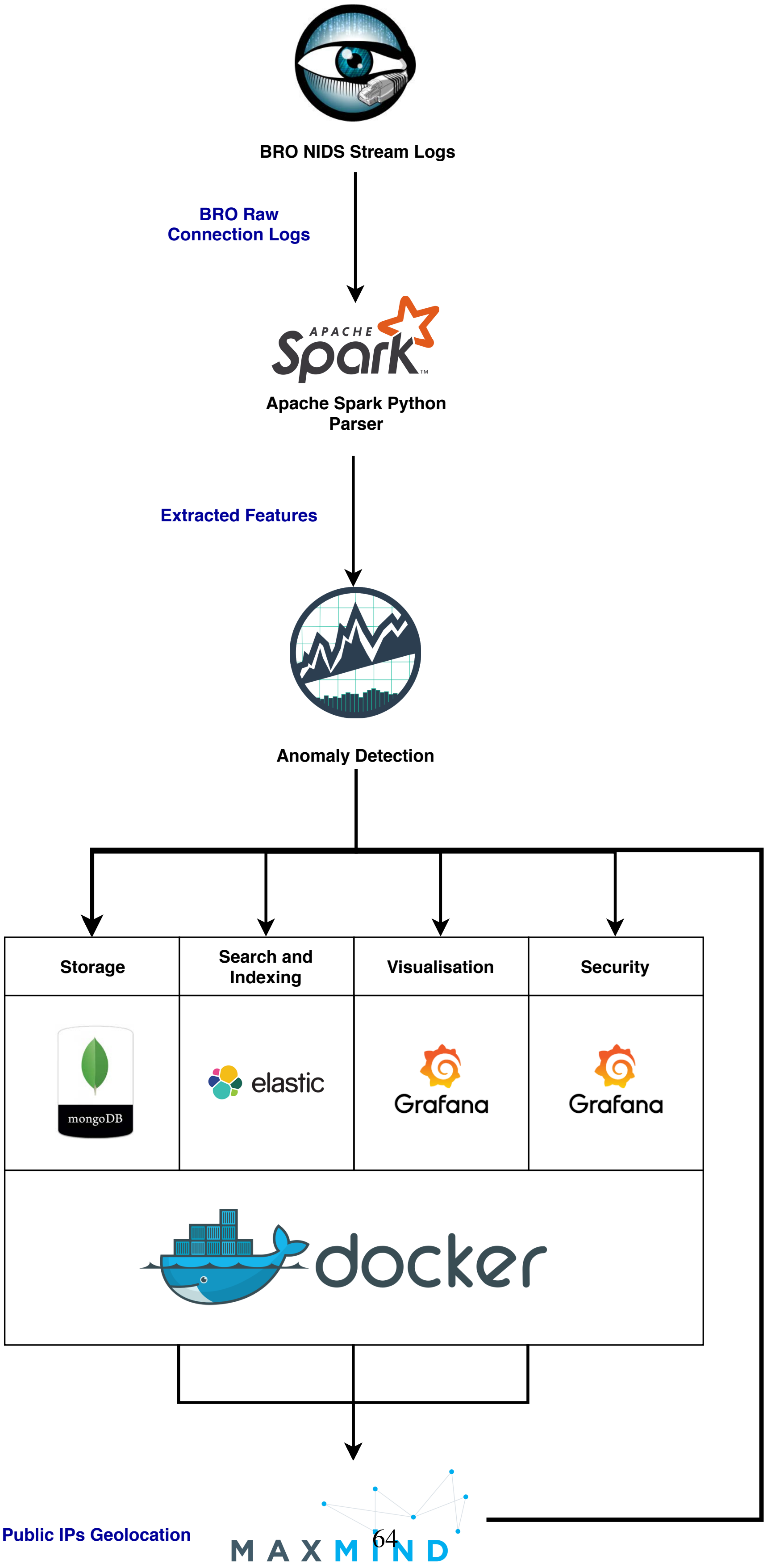


Figure 4.1: Daedalus Technical Overview

4.2 Evaluation

This section gives more details about our experimental dataset and the evaluation metrics used to validate our proposed technique.

4.2.1 Experimental Setup

All of our experiments are conducted on a dedicated processing server running CentOS Linux version 7 with Intel Xeon E5-2630 2.30GHz CPU and 126GB of RAM. Our framework is developed using Python programming language and by leveraging *Elasticsearch* as an indexing and search database for data analytics. *Apache Spark* alongside with *pandas* Python library are used to improve the scalability and to exploit the full capacity of our server (CPU and RAM resources). *Grafana* is used as a visualization tool on top of *Elasticsearch* in order to create the required dashboards. *Grafana* security functionalities are employed to provide additional functionalities of *user management*, *user authentication*, and *indices access roles*. All the data are stored in a storage server of 500TB capacity.

Experimental Dataset. In order to perform the experiments, a number of organizations participated to collect and share their IDS logs. Almost 90% of the participants installed BRO IDS, with some exceptions that employed other types of IDS. Moreover, the locations of the sensors were different. Some organizations choose to install their BRO sensors outside their firewalls, and therefore collect all the connections incoming and outgoing to/from their network. This results into sharing more consistent sizes of logs per day (sometimes hundreds of millions of connections each day). Whilst others capture only the traffic outgoing from their network, and thus less connection logs are received. In total, we receive millions of connection logs from the participants on a daily basis, which are stored on the dedicated storage server as compressed files.

Evaluation Metrics. In order to evaluate the performance and validate our approach, *Accuracy* ($\frac{TP+TN}{TP+FP+TN+FN}$), and F_1 *score* ($\frac{2*TP}{2*TP+FP+FN}$) metrics are computed.

4.2.2 Ground Truth Correlation

In this section, we validate the results of our anomaly detection technique by correlating them with some existing resources. We test our technique on a set of connection logs and identify the anomalies. Then, all the detected anomalous connections are correlated with external security feeds, such as malware sandboxing reports. The IP blocks are extracted from each connection (originator and responder IPs) and are correlated with the extracted malicious IPs from existing security sources within the same time interval that the connection occurred. We consider only three days of lag. Therefore, we first perform malware correlation within the same day that the connection occurred. If there is no match, we continue the correlation with the data of previous day. This process is repeated until the last three days. The reason of considering the three days of lag is due to the fact that most of the malware tend to change their IP addresses in order to evade detection. We further perform the correlation with an open-source ransomware database, *Ransomware Tracker* [9], which will be updated in the case of the detection of a new worldwide ransomware. Therefore, we download the *Ransomware Tracker* database, and on a weekly basis we check for any changes to keep our database updated.

From the correlation results, we consider the number of connections that were detected both by our system as anomalies and by the correlation as malicious as *true positives (TP)*, the ones that we flagged as anomalous but did not return correlation results as *false positives (FP)*, the benign connection that were not flagged as anomalies as *true negatives (TN)*, and the malicious connection that were not flagged as anomalous as *false negatives (FN)*. As an example, we examine $2,438,294$ connections detected as anomalous (from approximately 400 million connections) from one of the randomly chosen participating organization collected from August 20, 2018 to 4^{th} of September 2018. We correlate our results with both malware and ransomware databases and then measure the aforementioned security metrics. The obtained results show that we achieve 83.68% of accuracy and 91.11% of f1_score.

4.2.3 Validation on Benchmark Dataset

We further examine our proposed technique with benchmark datasets. We choose *UNBCIC 2017 IDS Dataset* [73, 7], which contains the BRO connection logs as well as two csv files that label the connections as Benign, DDoS or PortScan. We run our proposed technique on $350,137$ randomly selected connection logs in order to detect anomalies, among which $256,540$ were detected as true positives (flagged as anomalous and labeled either as DDoS attacks or Port Scan attacks), $16,775$ connections were detected as false positives (flagged as anomalous but there were no correlation results with DDoS and Port Scanning datasets),

$64,575$ were detected as true negatives (neither detected as anomalous nor DDoS nor Port Scanning), and $12,247$ were detected as false negatives (were not flagged as anomalous but there were correlation matches with DDoS and Port Scanning). The obtained results show that we achieve 91.71% of accuracy and 94.64% of f1_score.

4.2.4 Comparison with other Anomaly Detection Approaches

In this section, we compare our proposed technique with an exiting anomaly detection approach. To this end, we choose the *K-Means* clustering method [4] written in *Python* to detect anomalies on time series data. In this approach, the anomalies are detected based on the *reconstruction error curve*. We modify the provided implementation in order to fit it with the format of our datasets by using only four features (originator bytes and packets, and responder bytes and packets), and we also reduce the default number of clusters to 3. The implementation is applied on the same dataset of *UNBCIC 2017 IDS*, and then the results are correlated with the labelled dataset (DDoS and PortScan attacks).The obtained results show that we achieve 44.38% of accuracy and 56.28% of f1_score.

4.2.4.1 Long-Short Term Memory Network

In this section, we compare our proposed technique with LSTM based anomaly detection technique. To this end, we chose *keras-anomaly-detection* implementation proposed in [24] using *Python* programming language. The *Github* repository of the author contains several implementations in which one of them combines LSTMs and *Autoencoders* (*recurrent.py*) in order to detect time windows in time-series that contain anomalous patterns. We have changed the implementation in order to fit with the labeled connections of *UNBCIC 2017 IDS* and also reduced the number of batches to 4 due to scalability issues (the model training phase takes too much time). Finally, we compute the same evaluation metrics on the resulting classifications and we acheived 55.01% of accuracy and 67.99% of f1_score.

4.2.5 Scalability Study

We compare the execution time and resources consumption of our last two evaluations (subsections 4.2.3 and **??** using $350,137$ connections from *UNBCIC 2017 IDS*) with the K-Means algorithm and report the results as illustrated in Table 4.1. The *htop Linux* command [2] is used to monitor the percentage of each CPU core usage.

Table 4.1: Scalability Comparison

	Our Approach	**K-Means**
Execution Time	$19(m)$ and $25(s)$	1 hours and $16(m)$
CPU Cores Used	24 Cores Dedicated; less than 50% usage for each core	24 Cores Dedicated; 3 cores were used critically (more than 70% usage)
Memory(RAM) Used	4GB	5.22GB

4.3 Front-End Platform

This section gives a detailed description of the front-end platform that we have created for our security partners.

4.3.1 Dashboard Organisation

All the dashboards that we give access to our security partners are designed and organized the same way. Since each connection has two IP addresses (originator and responder IPs), we have decided to split the dashboards into mainly two columns: one for originator IPs information and another one with responder IPs information, in addition to some statistical curves. First, we display a time-series curve, which shows the unique count of these two IPs over the time of our analysis. This can give us an overview of the network traffic state over time and shows us the average number of destination IPs (responder IPs) to which originator IPs (source IPs) are connecting to as shown in Figure 4.2.

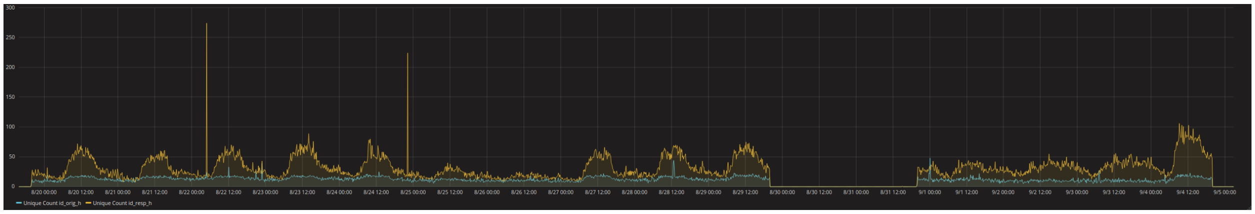
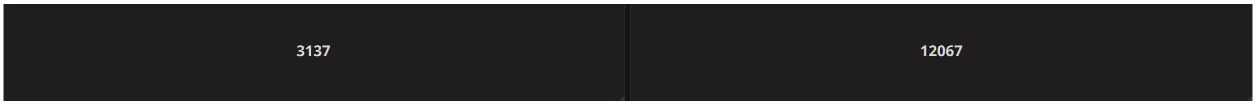


Figure 4.2: Time Series Overview

In addition to these time-series, we also show the same information aggregated in a single numerical value for both ends (originator and responder IPs), as shown in Figure 4.3.



3137

12067

Figure 4.3: Originator and Responder IPs Unique Count

After that, we display time series showing the statistical overview of attacks detected; from malware correlation phase 4.4, from ransomware correlation phase 4.5, and anomalies detected by our technique 4.6

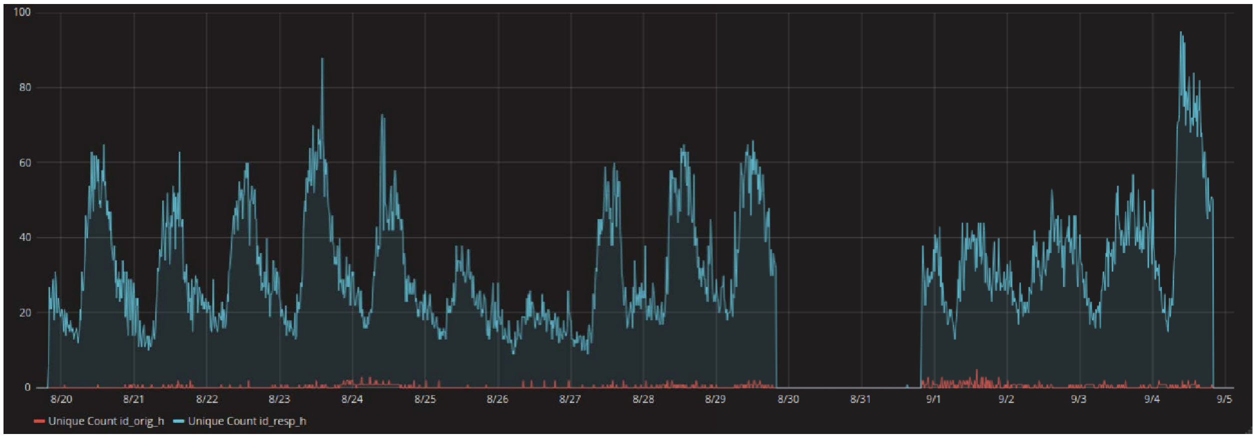


Figure 4.4: Malware Correlation Time Series

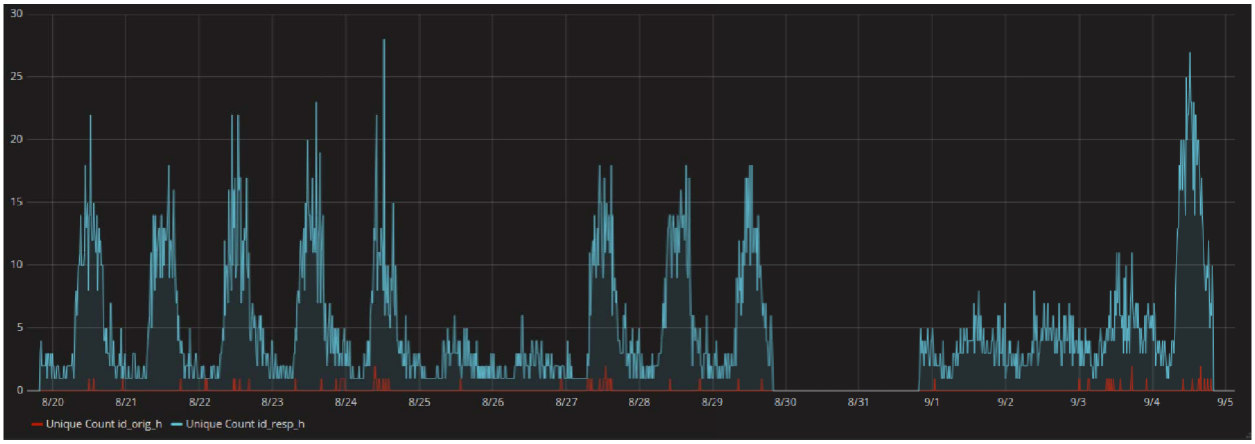


Figure 4.5: Ransomware Correlation Time Series

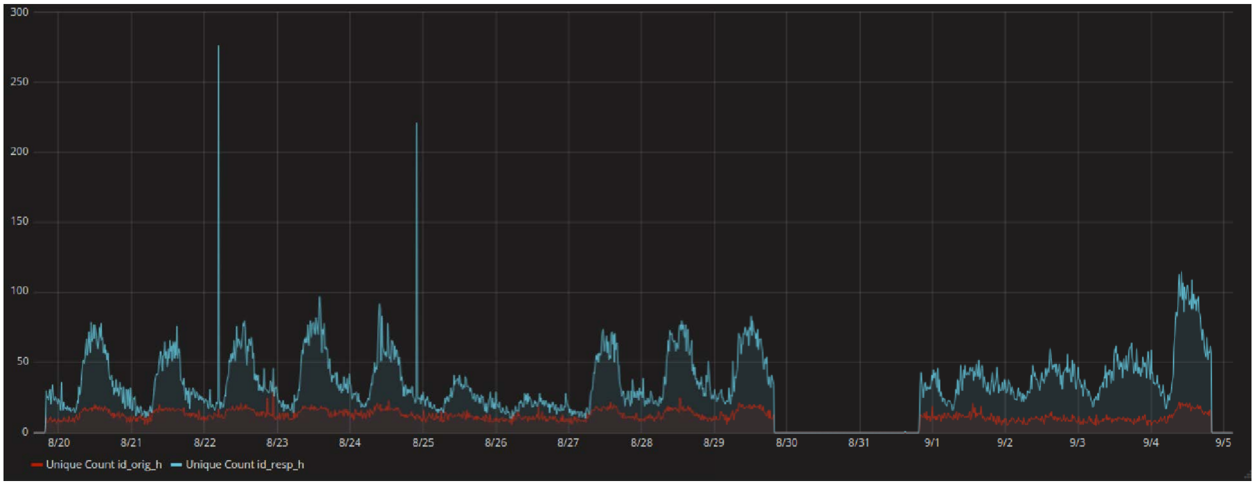


Figure 4.6: Anomalies Time Series

Then, we display geo-maps for both geolocated public originator and responder IPs, as shown in Figure 4.7

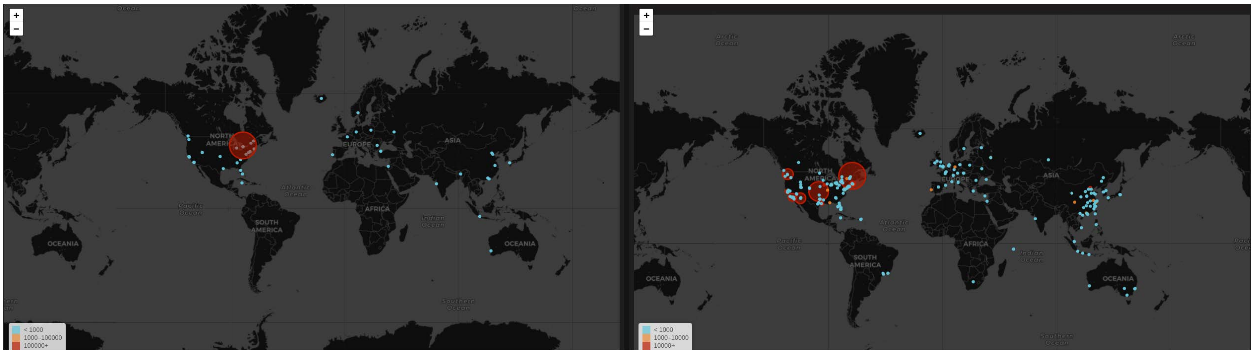
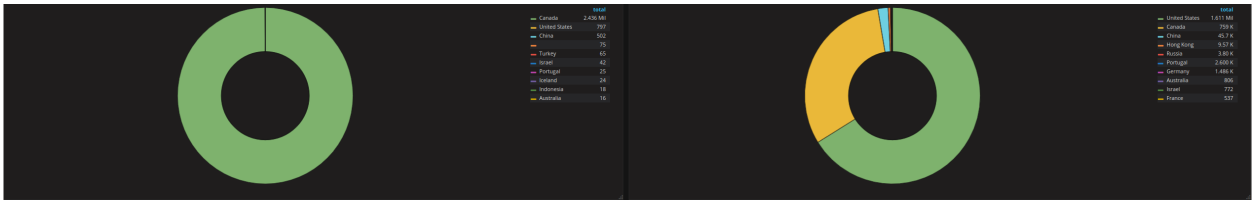


Figure 4.7: Originator and Responder IPs Geo-Maps

Then, we display two pie charts that show the top 10 countries in which these potentially malicious IPs (both originator and responder) are located in, as shown in Figure 4.8



	total
Canada	2.436 Mil
United States	797
China	502
	75
Turkey	65
Israel	42
Portugal	25
Iceland	24
Indonesia	18
Australia	16

	total
United States	1.611 Mil
Canada	759 K
China	45.7 K
Hong Kong	9.57 K
Russia	3.80 K
Portugal	2.600 K
Germany	1.486 K
Australia	806
Israel	772
France	537

Figure 4.8: Top 10 Originator and Responder Countries

Moreover, we also show information about the correlation results with the external feeds (malware and ransomware), as shown in Figure 4.9 (both originator and responder detected malware families) and Figure 4.10 (both originator and responder ransomware family and site type) .

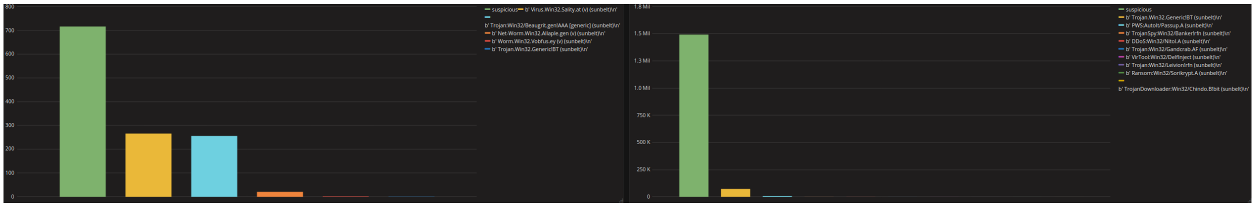


Figure 4.9: Top 10 Originator and Responder Malware Families

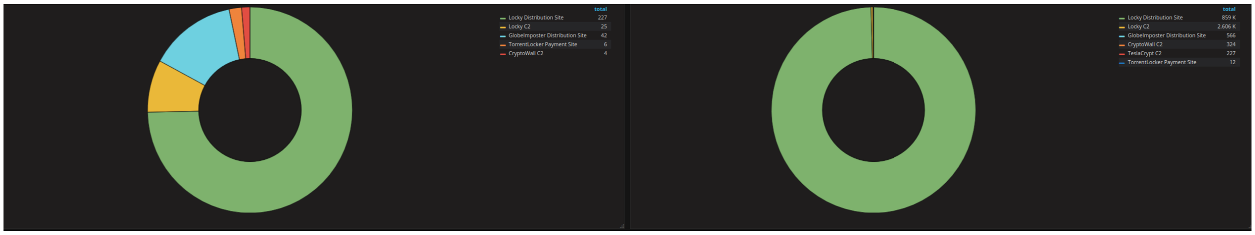


Figure 4.10: Top 10 Originator and Responder Ransomware Families

Chapter 5

Conclusion

Network anomaly detection is a hot research topic that is still evolving and attracting more researchers due to its importance in detecting security threats and its importance in analysing logged network traffic using IDS tools. In this research work, we presented *Daedalus*, a highly scalable time-series and unsupervised anomaly detection approach on massive connections data streams. *Daedalus* achieves our objectives by generating the maximum amount of cyber threat intelligence in an efficient and scalable manner, correlating this generated cyber threat intelligence with factual security sources of indicators of compromise (IoCs) in order to extend this intelligence with more security features and details, and validate the anomaly detection technique through extensive experimentation on real-life connections stream logs. We first presented the state of the art of existing network anomaly detection techniques. We analyzed these already existing works and found that the majority of them concentrate their work and validation on small real datasets or synthetic (benchmark) datasets. Therefore, their scalabilty results are not convenient and relevant for our case where we receive massive amounts of connections stream logs. Thus, we proposed a scalable time-series analysis technique that detects anomalies on these received logs in reasonable time and without consuming much resources (CPU and RAM). Consequently, the tool was developed using Python programming language alongside with some well-known state of the art tools and frameworks like Apache Spark for fast computations, Elasticsearch, mongoDB for storage purposes, Grafana for visualisation functionalities and to enable secure access to our front-end. Finally, we evaluated the accuracy and performance of our proposed technique during a three step process; ground truth correlation where we correlate the results of our algorithm with factual and reliable security feeds (malware and ransomware), validation on benchmark dataset where we apply our methodology on a synthetic well-known dataset (from UNBCIC), and comparison with other anomaly detection techniques where we compare the accuracy and performance results of our technique during the second step (on UNBCIC benchmark dataset) with other already existing implementations (K-Means and LSTMs). Subsequently, we have found that not only our algorithm achieves the highest accuracy results, but it also outperforms the other techniques in terms of execution times, CPU cores usage, and RAM usage. After achieving the aforementioned objectives and solving the stated problem, our research contributions consist of the following:

a) Comparative study of the existing state of the art network anomaly detection techniques.

b) Design and elaboration of a novel efficient and scalable network anomaly detection technique on massive connections data steams.

c) Design and implementation of a framework that analyzes massive connections data streams.

Daedalus can be improved in the future from different aspects. First, there is the issue of high rates of false positives on the real datasets (stream of BRO NIDS connection logs) which is normal due to the fact that BRO NIDS is characterized by its high false positives rate, but can be reduced by tunning our proposed algorithm (one possible solution is to change the parameters for the adaptive threshold). Second, the geolocation process is done on each IP address separately which consumes a lot of resources (time and CPU), one way to fix this is to summarize these IPs by ranges and process them.

Bibliography

[1] The bro network security monitor. https://www.bro.org/. [Online; accessed October-2017].

[2] htop(1) - linux man page. https://linux.die.net/man/1/htop.

[3] The bro network security monitor. https://www.zeek.org/bro-workshop-2011/slides/broverview.pdf, 2011. [Online; accessed March-2018].

[4] Anomaly detection with k-means clustering. http://amid.fish/anomaly-detection-with-k-means-clustering, 2015. [Online; accessed March-2015].

[5] How to check if time series data is stationary with python. https://machinelearningmastery.com/time-series-data-stationary-python/, 2016. [Online; accessed July-2016].

[6] An exponentially weighted moving average implementation that decays based on the elapsed time since the last update, approximating a time windowed moving average. https://gist.github.com/jhalterman/f7b18b30160ae7817bb93894056eb380, 2017.

[7] UNBCIC 2017 IDS Dataset. http://www.unb.ca/cic/datasets/ids-2017.html, 2017. [Online; accessed May-2018].

[8] Exploring the exponentially weighted moving average. https://www.investopedia.com/articles/07/ewma.asp, 2018. [Online; accessed August-2018].

[9] Ransomware tracker website. https://ransomwaretracker.abuse.ch/tracker/, 2018. [Online; accessed July-2018].

[10] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19 – 31, 2016.

[11] Tamer Aldwairi, Dilina Perera, and Mark A Novotny. An evaluation of the performance of restricted boltzmann machines as a model for anomaly network intrusion detection. *Computer Networks*, 144:111–119, 2018.

[12] Khaled Alrawashdeh and Carla Purdy. Toward an online anomaly intrusion detection system based on deep learning. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pages 195–200. IEEE, 2016.

[13] Fatemeh Amiri, MohammadMahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *Journal of Network and Computer Applications*, 34(4):1184–1199, 2011.

[14] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2:1–18, 2015.

[15] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Security Symposium*, 2017.

[16] Monowar H Bhuyan, DK Bhattacharyya, and Jugal K Kalita. A multi-step outlier-based anomaly detection approach to network-wide traffic. *Information Sciences*, 348:243–271, 2016.

[17] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[18] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Ndss*, 2011.

[19] Loïc Bontemps, James McDermott, Nhien-An Le-Khac, et al. Collective anomaly detection based on long short-term memory recurrent neural networks. In *International Conference on Future Data and Security Engineering*, pages 141–152. Springer, 2016.

[20] Yacine Bouzida, Frederic Cuppens, Nora Cuppens-Boulahia, and Sylvain Gombault. Efficient intrusion detection using principal component analysis. In *3éme Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, France*, pages 381–395, 2004.

[21] José Camacho, Alejandro Pérez-Villegas, Pedro García-Teodoro, and Gabriel Maciá-Fernández. Pca-based multivariate statistical network monitoring for anomaly detection. *Computers & Security*, 59:118–137, 2016.

[22] James Cannady. Artificial neural networks for misuse detection. In *National information systems security conference*, volume 26. Baltimore, 1998.

[23] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. Host based intrusion detection system with combined cnn/rnn model. In *Proceedings of Second International Workshop on AI in Security*, 2018.

[24] Xianshun Chen. keras-anomaly-detection. https://github.com/chen0040/keras-anomaly-detection, 2018. [Online; accessed July-2018].

[25] Abebe Diro and Naveen Chilamkurti. Leveraging lstm networks for attack detection in fog-to-things communications. *IEEE Communications Magazine*, 56(9):124–130, 2018.

[26] Holger Dreger, Christian Kreibich, Vern Paxson, and Robin Sommer. Enhancing the accuracy of network-based intrusion detection with host-based context. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 206–221. Springer, 2005.

[27] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[28] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[29] Fahimeh Farahnakian and Jukka Heikkonen. A deep auto-encoder based approach for intrusion detection system. In *Advanced Communication Technology (ICACT), 2018 20th International Conference on*, pages 178–183. IEEE, 2018.

[30] Cheng Feng, Tingting Li, and Deeph Chana. Multi-level anomaly detection in industrial control systems via package signatures and lstm networks. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 261–272. IEEE, 2017.

[31] Gilberto Fernandes Jr, Luiz F Carvalho, Joel JPC Rodrigues, and Mario Lemes Proença Jr. Network anomaly detection using ip flows with principal component analysis and ant colony optimization. *Journal of Network and Computer Applications*, 64:1–11, 2016.

[32] Ni Gao, Ling Gao, Quanli Gao, and Hai Wang. An intrusion detection model based on deep belief networks. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, pages 247–252. IEEE, 2014.

[33] Farnaz Gharibian and Ali A Ghorbani. Comparative study of supervised machine learning techniques for intrusion detection. In *Communication Networks and Services Research, 2007. CNSR'07. Fifth Annual Conference on*, pages 350–358. IEEE, 2007.

[34] David Goldberg and Yinan Shan. The importance of features for statistical anomaly detection. In *HotCloud*, 2015.

[35] Arnaldo Gouveia and Miguel Correia. A systematic approach for the application of restricted boltzmann machines in network intrusion detection. In *International Work-Conference on Artificial Neural Networks*, pages 432–446. Springer, 2017.

[36] Anderson Hiroshi Hamamoto, Luiz Fernando Carvalho, Lucas Dias Hiera Sampaio, Taufik Abrão, and Mario Lemes Proença Jr. Network anomaly detection system using genetic algorithm and fuzzy logic. *Expert Systems with Applications*, 92:390–402, 2018.

[37] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

[38] Wenjie Hu, Yihua Liao, and V Rao Vemuri. Robust support vector machines for anomaly detection in computer security. In *ICMLA*, pages 168–174, 2003.

[39] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[40] Amin Karami and Manel Guerrero-Zapata. A fuzzy anomaly detection system based on hybrid pso-kmeans algorithm in content-centric networks. *Neurocomputing*, 149:1253–1269, 2015.

[41] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon. Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. *arXiv preprint arXiv:1611.01726*, 2016.

[42] Jihyun Kim, Howon Kim, et al. An effective intrusion detection classifier using long short-term memory with gradient descent optimization. In *Platform Technology and Service (PlatCon), 2017 International Conference on*, pages 1–6. IEEE, 2017.

[43] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *Platform Technology and Service (PlatCon), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[44] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 173–191. Springer, 2003.

[45] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C Suh, Ikkyun Kim, and Kuinam J Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, pages 1–13, 2017.

[46] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[47] Chaopeng Li, Jinlin Wang, and Xiaozhou Ye. Using a recurrent neural network and restricted boltzmann machines for malicious traffic detection. *NeuroQuantology*, 16(5), 2018.

[48] Lianpeng Li, Lun Xie, Weize Li, Zhenzong Liu, and Zhiliang Wang. Improved deep belief networks (idbn) dynamic model-based detection and mitigation for targeted attacks on heavy-duty robots. *Applied Sciences (2076-3417)*, 8(5), 2018.

[49] Yang Li and Li Guo. An active learning based tcm-knn algorithm for supervised network intrusion detection. *Computers & security*, 26(7-8):459–467, 2007.

[50] Yichen Li, Yinghua Ma, Mingda Guo, and Shenghong Li. Anomaly detection for power grid flow patterns based on the multi-restricted boltzmann machines. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 628–633. IEEE, 2018.

[51] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424–430, 2012.

[52] Yuancheng Li, Rong Ma, and Runhai Jiao. A hybrid malicious code detection method based on deep learning. *methods*, 9(5), 2015.

[53] Yihua Liao and V Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection1. *Computers & security*, 21(5):439–448, 2002.

[54] Richard P Lippmann and Robert K Cunningham. Improving intrusion detection performance using keyword selection and neural networks. *Computer networks*, 34(4):597–603, 2000.

[55] Tao Ma, Fen Wang, Jianjun Cheng, Yang Yu, and Xiaoyun Chen. A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. *Sensors*, 16(10):1701, 2016.

[56] Pheeha Machaka, Antoine Bagula, and Fulufhelo Nelwamondo. Using exponentially weighted moving average algorithm to defend against ddos attacks. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–6. IEEE, 2016.

[57] Erik Marchi, Fabio Vesperini, Florian Eyben, Stefano Squartini, and Björn Schuller. A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional lstm neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1996–2000. IEEE, 2015.

[58] Federico Marini and Beata Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.

[59] Jerry M Mendel. Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE*, 83(3):345–377, 1995.

[60] Soosan Naderi Mighan and Mohsen Kahani. Deep learning based latent feature extraction for intrusion detection. In *Electrical Engineering (ICEE), Iranian Conference on*, pages 1511–1516. IEEE, 2018.

[61] Ali H Mirza and Selin Cosan. Computer network intrusion detection using sequential lstm neural networks autoencoders. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2018.

[62] Rizwan Mushtaq. Augmented dickey fuller test. 2011.

[63] Miguel Nicolau, James McDermott, et al. A hybrid autoencoder and density estimation model for anomaly detection. In *International Conference on Parallel Problem Solving from Nature*, pages 717–726. Springer, 2016.

[64] The Fifth International Conference on Knowledge Discovery and Data Mining. Kdd cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. [Online; accessed July-2017].

[65] Chandrika Palagiri. Network-based intrusion detection using neural networks. *epartment of Computer Science Rensselaer Polytechnic Institute Troy, New York*, pages 12180–3590, 2002.

[66] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[67] G Poojitha, K Naveen Kumar, and P Jayarami Reddy. Intrusion detection using artificial neural network. In *Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on*, pages 1–7. IEEE, 2010.

[68] Feng Qu, Jitao Zhang, Zetian Shao, and Shuzhuang Qi. An intrusion detection model based on deep belief network. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, pages 97–101. ACM, 2017.

[69] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Distributed anomaly detection in wireless sensor networks. In *Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on*, pages 1–5. IEEE, 2006.

[70] Manikantan Ramadas, Shawn Ostermann, and Brett Tjaden. Detecting anomalous network traffic with self-organizing maps. In *International Workshop on Recent Advances in Intrusion Detection*, pages 36–54. Springer, 2003.

[71] Sanjiban Sekhar Roy, Abhinav Mallik, Rishab Gulati, Mohammad S Obaidat, and PV Krishna. A deep learning based artificial neural network approach for intrusion detection. In *International Conference on Mathematics and Computing*, pages 44–53. Springer, 2017.

[72] Mateu Sbert, Han-Wei Shen, Ivan Viola, Min Chen, Anton Bardera, and Miquel Feixas. Tutorial on information theory in visualization. In *SIGGRAPH Asia 2017 Courses*, page 17. ACM, 2017.

[73] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[74] Rhythima Shinde, Pieter Van der Veeken, Stijn Van Schooten, Jan van den Berg, Bishnu Giri, Avinashkumar Bhardwaj, G. R. K. S. Subrahmanyam, and Vinay Avasthi. Survey on ransomware: A new era of cyber attack. 2017.

[75] Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799–3821, 2007.

[76] Robin Sommer. Bro: An open source network intrusion detection system. In *DFN-Arbeitstagung über Kommunikationsnetze*, pages 273–288, 2003.

[77] Ralf C Staudemeyer. Applying long short-term memory recurrent neural networks to intrusion detection. *South African Computer Journal*, 56(1):136–154, 2015.

[78] Ralf C Staudemeyer and Christian W Omlin. Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, pages 218–224. ACM, 2013.

[79] Iwan Syarif, Adam Prugel-Bennett, and Gary Wills. Data mining approaches for network intrusion detection: from dimensionality reduction to misuse and anomaly detection. *Journal of Information Technology Review*, 3(2):70–83, 2012.

[80] T Tang, Syed Ali Raza Zaidi, Des McLernon, Lotfi Mhamdi, and Mounir Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 IEEE International Conference on Network Softwarization (NetSoft 2018)*, pages 202–206. IEEE, 2018.

[81] Jingjing Tian and Ping'An Li. An intrusion detection algorithm of dynamic recursive deep belief networks. In *Proceedings of the 2017 International Conference on Information Technology*, pages 180–183. ACM, 2017.

[82] Cynthia Wagner, Jérôme François, Thomas Engel, et al. Machine learning approach for ip-flow record anomaly detection. In *International Conference on Research in Networking*, pages 28–39. Springer, 2011.

[83] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[84] Xiaodong Wang, Haoran Zhang, Changjiang Zhang, Xiushan Cai, Jinshan Wang, and Meiying Ye. Time series prediction using ls-svm with particle swarm optimization. In *International Symposium on Neural Networks*, pages 747–752. Springer, 2006.

[85] Jun Yang, Jiangdong Deng, Shujuan Li, and Yongle Hao. Improved traffic detection with support vector machine based on restricted boltzmann machine. *Soft Computing*, 21(11):3101–3112, 2017.

[86] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017.

[87] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5):649–659, 2008.

[88] Yin Zhang and Vern Paxson. Detecting stepping stones. In *USENIX Security Symposium*, volume 171, page 184, 2000.

[89] Zheng Zhang, Jun Li, CN Manikopoulos, Jay Jorgenson, and Jose Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proc. IEEE Workshop on Information Assurance and Security*, pages 85–90, 2001.

[90] Guangzhen Zhao, Cuixiao Zhang, and Lijuan Zheng. Intrusion detection using deep belief network and probabilistic neural network. In *Computational Science and Engineering (CSE) and Embedded and Ubiquitous Computing (EUC), 2017 IEEE International Conference on*, volume 1, pages 639–642. IEEE, 2017.