Designing an effective network forensic framework for the investigation of botnets in the Internet of Things

Nickolaos Koroniotis

A thesis submitted in fulfilment of the requirements

for the degree of

Doctor of Philosophy

School of Engineering and Information Technology

The University of New South Wales

Australia

March 2020

Thesis Dissertation Sheet

Surname/Family Name	:	Koroniotis
Given Name/s	:	Nickolaos
Abbreviation for degree as give in the University calendar	:	PhD
Faculty	:	UNSW Canberra at ADFA
School	:	UC Engineering & Info Tech
Thesis Title	:	Designing an effective network forensic framework for the investigation of botnets in the Internet of Things

Abstract 350 words maximum:

The emergence of the Internet of Things (IoT), has heralded a new attack surface, where attackers exploit the security weaknesses inherent in smart things. Comprised of heterogeneous technologies and protocols, the IoT is a source of high-speed and volume data, rendering pre-existing forensic solutions ineffective. As a result, developing new network forensic solutions for the IoT is imperative. Some of the challenges involved in designing network forensic solutions for the IoT are 1) obtaining realistic data that represent contemporary network behaviour, 2) selecting and optimizing a machine learning model, best suited to deal with such data and 3) identifying and tracing attacks. This thesis provides considerable contribution to the research focusing on building a network forensic framework tasked with investigating botnet activities in IoT networks.

The first contribution is the design of a new virtual testbed and the generation of a new network dataset, called Bot-IoT. This new dataset incorporates normal IoT traffic and represents a range of realistic network attacks. The second contribution is the selection of optimal features for the dataset. The process combined two measures, namely Pearson Correlation and Joint Entropy to create a score for the features, allowing for the selection of the 10 least-similar, which helped in removing any redundant information from the dataset.

The third contribution is the analysis performed on the Bot-IoT dataset. For this analysis, two other widely used dataset, the UNSW-NB15 and NSL-KDD datasets were selected and seven machine learning models were trained. The fourth contribution is the development of the Particle Deep Framework (PDF) which covers the stages of the digital forensic investigation process. The PDF utilizes Particle Swarm Optimization for the selection of the optimal hyperparameters of a deep learning model, which lies at its core and is trained to detect attack network flows.

Originality Statement

Inclusion of Publicaitons Statement

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in their thesis in lieu of a Chapter if:

- The candidate contributed greater than 50% of the content in the publication and is the “primary author”, ie. the candidate was responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has approval to include the publication in their thesis in lieu of a Chapter from their supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis

Please indicate whether this thesis contains published material or not:

☐ This thesis contains no publications, either published or submitted for publication.

☐ Some of the work described in this thesis has been published and it has been documented in the relevant Chapters with acknowledgement.

☒ This thesis has publications (either published or submitted for publication) incorporated into it in lieu of a chapter and the details are presented below.

CANDIDATE’S DECLARATION I declare that: • I have complied with the UNSW Thesis Examination Procedure • where I have used a publication in lieu of a Chapter, the listed publication(s) below meet(s) the requirements to be included in the thesis.		
Candidate’s Name	Signature	Date (dd/mm/yy)

POSTGRADUATE COORDINATOR’S DECLARATION I declare that: • the information below is accurate • where listed publication(s) have been used in lieu of Chapter(s), their use complies with the UNSW Thesis Examination Procedure • the minimum requirements for the format of the thesis have been met.		
PGC’s Name	PGC’s Signature	Date (dd/mm/yy)

Details of Publicaiton #1: Full title: *Forensics and deep learning mechanisms for botnets in Internet of Things: A survey of challenges and solutions* Authors: *Koroniotis, Nickolaos and Moustafa, Nour and Sitnikova, Elena* Journal or book name: *IEEE Access* Volume/page numbers: *7/61764–61785* Date accepted/published: *14 May 2019*						
Status	Published	☒	Accepted and in press	☐	In progress (Submitted)	☐
The Candidate's Contribution to the Work The candidate designed, implemented and is the primary author of this work.						
Location of the work in the thesis and/or how the work is incorporated in the thesis: The publication was used in lieu of Chapter 2, as the Literature Review.						
PRIMARY SUPERVISOR'S DECLARATION						
I declare that: • the information above is accurate • this has been discussed with the PGC and it is agreed that this publication can be included in this thesis in lieu of a Chapter • All of the co-authors of the publication have reviewed the above information and have agreed to its veracity by signing a 'Co-Author Authorisation' form.						
Primary Supervisor's name		Primary Supervisor's signature			Date (dd/mm/yy)	

Details of Publicaiton #2: Full title: *Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset* Authors: *Koroniotis, Nickolaos and Moustafa, Nour and Sitnikova, Elena and Turnbull, Benjamin* Journal or book name: *Future Generation Computer Systems* Volume/page numbers: *100/779–796* Date accepted/published: *15 May 2019*						
Status	Published	☒	Accepted and in press	☐	In progress (Submitted)	☐
The Candidate's Contribution to the Work The candidate designed, implemented and is the primary author of this work.						
Location of the work in the thesis and/or how the work is incorporated in the thesis: The publication was used in lieu of Chapter 3, to describe the design, generation and refinement of the Bot-IoT dataset.						
PRIMARY SUPERVISOR'S DECLARATION						
I declare that: • the information above is accurate • this has been discussed with the PGC and it is agreed that this publication can be included in this thesis in lieu of a Chapter • All of the co-authors of the publication have reviewed the above information and have agreed to its veracity by signing a 'Co-Author Authorisation' form.						
Primary Supervisor's name			Primary Supervisor's signature			Date (dd/mm/yy)

Details of Publicaiton #3: Full title: *A New Network Forensic Framework based on Deep Learning for Internet of Things Networks: A Particle Deep Framework* Authors: *Koroniotis, Nickolaos and Moustafa, Nour and Sitnikova, Elena* Journal or book name: *Future Generation Computer Systems* Volume/page numbers: *110/91-106* Date accepted/published: *30 March 2020*						
Status	Published	☒	Accepted and in press	☐	In progress (Submitted)	☐
The Candidate's Contribution to the Work The candidate designed, implemented and is the primary author of this work.						
Location of the work in the thesis and/or how the work is incorporated in the thesis: The publication was used in lieu of Chapter 5, to describe the design of the Particle Deep Framework.						
PRIMARY SUPERVISOR'S DECLARATION						
I declare that: • the information above is accurate • this has been discussed with the PGC and it is agreed that this publication can be included in this thesis in lieu of a Chapter • All of the co-authors of the publication have reviewed the above information and have agreed to its veracity by signing a 'Co-Author Authorisation' form.						
Primary Supervisor's name		Primary Supervisor's signature			Date (dd/mm/yy)	

Acknowledgments

I would like to express my gratitude to my research supervisors, Dr Elena Sitnikova, Dr Nour Moustafa and my external supervisor Professor Jill Slay. Their patient guidance, constructive comments and support allowed me to persevere and better shape this research, while at the same time allowing me to learn much from them. I would also like to thank the Commonwealth, for providing me with a Scholarship that allowed me to study and live away from home, and thus, in a sense, enabled this research. I would also like to thank Dr Benjamin Turnbull and Mr Rory Macleod for giving me access to the Cyber Range and helping me port the environment that was used to generate the dataset which enabled the construction of the framework. Last but certainly not least, I would like to thank my friends and family, namely my parents Aristotelis Koroniotis and Eugenia Koroniotis-Papadopoulos, my brother Ioannis Koroniotis and my grandparents Plato and Penelope Koroniotis and Ioannis and Efrosini Papadopoulos. They helped me pull through one of the toughest experiences in my life and I will forever be in their debt.

Abstract

The emergence of the Internet of Things (IoT), has heralded new attack surfaces, where attackers exploit the security weaknesses inherent in smart things. The IoT is comprised of heterogeneous devices and protocols which is a source of high-speed and volume data, rendering pre-existing forensic solutions ineffective. As a result, developing new network forensic solutions for the IoT is imperative. The key challenges involved in designing network forensic solutions for the IoT include: 1) obtaining realistic data that represent contemporary network behaviour, 2) selecting and optimizing a machine learning model, best suited to deal with such data and 3) identifying and tracing attacks. This thesis provides a considerable contribution to the research focusing on building a network forensic framework tasked with investigating botnet activities in IoT networks.

The first contribution is the design of a new virtual IoT network testbed and the generation of a new network dataset, called Bot-IoT. This new dataset incorporates normal IoT traffic and represents a range of realistic network attacks. This dataset has new IoT features that do not exist in the literature, along with new security events of botnets, for evaluating new network forensics and intrusion detection systems. The second contribution is the selection of optimal features that can be used to build effective network forensics techniques based on machine learning. The process combined two measures, namely Pearson Correlation and Joint Entropy to create a score for the features, allowing for selecting the most important features and improving the techniques' performances. This helped in removing any redundant information from the dataset.

The third contribution is the analysis performed on the Bot-IoT dataset. For this analysis, two other widely used dataset, the UNSW-NB15 and NSL-KDD datasets were selected and seven machine learning models were trained and validated to demonstrate how forensics techniques can be developed using machine learning rather than traditional forensics tools that can not trace new attack families. The fourth contribution is the development of novel Particle Deep Framework (PDF) which covers the stages of the digital forensic investigation process. The

PDF utilizes Particle Swarm Optimization for the selection of the optimal hyperparameters of a deep learning model, which automates its potential process of learning botnet events, and correctly trace their anomalous behaviours in IoT networks.

Keywords

Botnet, Internet of Things, Network Forensics, Neural Network, Multi-layer perceptron, Partical Swarm Optimization

List of Terms/Abbreviations

Acronyms	Description
ANN	Artificial Neural Network
Bot-IoT	Botnet Internet of Things
CAIDA	Center of Applied Internet Data Analysis
C&C	Command and Control
CIA	Confidentiality Integrity Availability
CICIDS	Canadian Institute of Cybersecurity Intrusion Detection System
CNN	Convolutional Neural Network
DAE	Deep Auto Encoder
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DDoS	Distributed Denial of Service
DEFCON	Defense readiness condition
DFRWS	Digital forensic research workshop
DoS	Denial of Service
DL	Deep Learning
DNN	Deep Neural Network
DPI	Deep Packet Inspection
GBps	Gigabytes per second
HTTP	Hypertext transfer protocol
IAB	Internet Architecture Board
ISCX	Information Security Center of Excelence
IDS	Intrusion Detection System
IoT	Internet of Things
IoTAA	Internet of Things Alliance Australia
IRC	Internet Relay Chats
LBNL	Lawrence Berkeley National Lab
ML	Machine Learning
MLP	Multi-layer perceptron

Acronyms	**Description**
MQTT	Message queue telemetry transport
NIDS	Network Intrusion Detection System
OS	Operating System
PC	Perconal Computer
Pcap	Packet capture
PCC	Pearson Correlation Coefficient
PDF	Particle Deep Framework
PSO	Particle Swarm Optimisation
ReLU	Rectified Linear Unit
RFID	Radio-frequency identificaiton
RNN	Recurrent Neural Network
SJE	Shannon Joint Entropy
SQL	Structure Query Language
SVM	Support Vector Machine
Tbps	Terabits per second
TCP	Transfer Control Protocol
TUIDS	Tezpur University Intrusion Detection System
UDP	User Datagram Protocol
UNIBS	University of Brescia
UNSW-NB15	University of New South Wales Network Based 2015
VM	Virtual Machine

List of publications

Journal Articles

- Koroniotis, N., Moustafa, N. and Sitnikova, E., 2019. Forensics and Deep Learning Mechanisms for Botnets in Internet of Things: A Survey of Challenges and Solutions. IEEE Access, 7, pp.61764-61785.
- Koroniotis, N., Moustafa, N., Sitnikova, E. and Turnbull, B., 2019. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. Future Generation Computer Systems, 100, pp.779-796.
- Koroniotis, N., Moustafa, N., Sitnikova, E., 2020. A New Network Forensic Framework based on Deep Learning for Internet of Things Networks: A Particle Deep Framework. Future Generation Computer Systems. (Second Round of Review)

Conference Papers

- Koroniotis, N., Moustafa, N., Sitnikova, E. and Slay, J., 2017, December. Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques. In International Conference on Mobile Networks and Management (pp. 30-44). Springer, Cham.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1. Overview of Cyber Security in the Internet of Things

In recent times, the world has come to rely greatly on the Internet and the services that it provides. As such, it should come as no surprise that innovations that harness the interconnectivity that the Internet offers, would themselves become integral parts of everyday life. Heterogeneous technologies and devices the era of “ubiquitous computing” lead to the emergence of the Internet of Things (IoT). IoT is one such innovation that has exhibited a dramatic growth over the years [5]. IoT systems that combine sensors, computation capabilities, networking, and actuators are also called cyber-physical systems. It enables various services, providing automation, cost-efficiency, and precision with implementations spanning multiple industrial sectors, including healthcare, water, power and agriculture [6]. As such, it is evident that both academic and industry institutions rely greatly on the IoT.

The number of deployed IoT devices has exploded in recent years. In 2018, approximately 7 billion IoT devices were connected to the Internet, while projections suggest that by 2020 they will reach 20.4 billion [7]. The most popular types of IoT installations in 2017 were smart home devices, like smart fridge, smart garage door, smart lighting, and air-conditioning at about 663 million units. IoT devices are designed to be constantly active and utilize a wireless network to exchange data and commands. Communication in an IoT network can either be in the form of thing-to-thing, where things communicate locally or thing-to-cloud, where devices collectively transmit sensor data to the backend cloud infrastructure and receive instructions [8]. While these devices provide important benefits such as optimization of business processes and centralized control of actuators distributed in diverse geographical locations, they are often targeted by cyber attacks due to their availability and vulnerable designs. According to a report by Symantec 2018 [9], the total number of attacks targeting IoT devices for that year exceeded 57,000, with more than 5,000 attacks being recorded each month.

Commonly, IoT devices are built with limited computing capabilities and constrained power supply. This is because they are often designed to be mobile, set-up in places where they can not be plugged into a power outlet or they are programmed to function in short periods of the day. Furthermore, it has been shown [10, 11] that a good portion of the online IoT devices exhibits a severe lack of security measures. Thus, the lack of security, coupled with the "always-active" design, has given an increase to several types of cyber attacks that either target or utilize IoT for further exploitation. These attacks vary from massive DDoS attacks that reach several hundred Gbps or even Tbps and can paralyze entire sectors of the Internet [12, 13], to the exploitation of cyber-physical systems which allows criminals to gain access to restricted areas [14], to utilizing compromised IoT devices to gain access to secured subnetworks. Therefore, as the risks and insecurities of the IoT persist, developing digital forensic methods becomes of utmost importance.

Since there exist no standards for designing and implementing the IoT is heterogenous, with multiple protocols and technologies often coexisting in a single system. As such, it is challenging to design digital forensic solutions that can be applied to multiple IoT systems. Digital forensics is the use of scientifically proven methods for the extraction, preservation, examination, analysis, and presentation of digital evidence obtained from cyberspace [15]. Along with the constant evolution of technology, so digital forensics should be evolved into several subdomains, such as network, cloud, mobile and IoT forensics [16]. In order to circumvent the complexity and heterogeneity, network forensics can be chosen to investigate IoT security incidents, as most implementations utilize the TCP/IP protocol suite for communications with the backend cloud platform. Many network forensic frameworks have been proposed, but for IoT, most of them focus on the acquisition phase [17, 18, 19, 20]. Thus, there is a need for a more complete network forensic framework that also incorporates the analysis and examination phases in their process.

The rest of this chapter is organised as follows. Forms of Cybercrime are discussed in Section 1.2; Section 1.3 describes the problem formulation and the research questions; Section 1.4 outlines the various types of cybercrime and a basic introduction to digital forensics. Finally, the thesis contributions are given in Section 1.5 and the thesis structure is given in Section 1.6.

1.2. Types of Cyber crime

A system is deemed secure if the key principles of the Confidentiality, Integrity and

Availability (CIA) triad are preserved [21]. Each principle of the CIA focuses on a different aspect of a system. Confidentiality dictates that information needs to be protected from unauthorized access, Integrity purports that data should not be either deleted or modified by unauthorized entities while Availability means that access to a user's data should not be hinder. Through careful planning, cyber-criminals are capable of bypassing an IoT system's security and its network, and in the process breach the CIA triad. The motive behind an attack can vary from simply compromising an IoT device and stealing information it has in its memory, to escalating and using the compromised IoT device as an attack vector to target other devices in the network.

Among the many threats that exist on the Internet, the ones best known for their diverse actions and destructive capabilities are botnets. Botnets are networks of infected devices, which are under the control of malicious individuals named botmasters and can be used to perform a number of different attacks, such as DDoS/DoS, identity theft, data exfiltration, e-mail spamming, keylogging and malware propagation [22]. Although originally targeting PCs, laptops, and servers, the always-online design of IoT devices coupled with their weak security measures have made them the main targets for botmasters, giving rise to a new generation of IoT-based botnets [12, 13]. IoT-based botnets have further expanded the repertoire of their actions, by including attacks that IoT devices are particularly susceptible towards, such as bricking and Sybil attacks [23]. Many botnets from this new generation made an appearance in the period 2017-2018, causing widespread damage to IoT devices and even disabling entire sectors of the Internet [12, 24, 25].

A prime example of IoT-based botnets was Mirai which made its appearance in 2016 and spawned many variants [12]. At its peak, Mirai was the cause of record-breaking DDoS attacks, peaking at 1.1 Tbps which was used to target On Vous Héberge (OVH) that is a French cloud service provider. Although Mirai was quite effective in infecting bots and attacking, its infrastructure was relatively simplistic. Thus, more sophisticated botnets appeared, like Malware-MustDie which was equipped with encryption modules for Command & Control (C&C) communication [12]. Persirai, a variant of Mirai which was discovered in April 2017 infected as many as 120,000 devices by using a zero-day exploit [12]. Hajime used a distributed C&C infrastructure, with communications encrypted using a public-private key scheme. BrickerBot was a particularly destructive botnet which leveraged default ssh credentials and attempted to permanently disable any infected IoT devices [24].

In addition to the common botnet attacks, IoT-based botnets are capable of performing a range of new attacks with an impact on cyber-physical systems. Bricking is one such attack, known as permanent-denial-of-service attack, which disables an IoT device by manipulating its firmware [24]. Malware propagation through proximity infection is another attack which was established by Ronen et al. [26]. In this scenario, adjacent IoT devices infect each other, spreading the malware binary from a single IoT device to an entire building or even a city. Finally, cyber-attacks can have effects in the physical world, with attackers being able to unlock doors, cause false alarms, enable/disable electronic devices, modify sensor data or even track the user, where all of which can be used to enable physical crimes like breaking into a smarthome and burglaring [14, 23, 27, 28].

The threat caused by cyber-attacks is constantly on the rise. According to a 2019 report by Akamai [29], more than 18 million malicious logins originated from the US, 5 million from Russia and just over 3 million from Brazil between 2017-2019. In a report by McAfee [30], for the first three quarters of 2018, over 100,000 new IoT-based malware were discovered. Therefore, there is indeed a serious motivation to build effective network forensic solutions for investigating botnets in IoT settings.

1.3. Problem formulation and research Questions

In this PhD, we tackle the research problem of identifying and tracing new botnet attack families from large-scale IoT networks. Network flows, which are a statistical representation of the flows in a network, were chosen as the underline data-source used by the framework. The expected framework that should address this problem, needs to be able to distinguish effectively between normal and abnormal flows, in a timely fashion.

The problem is separated into the following three major sub-problems.

- **Sub-problem 1**: detecting sophisticated attack scenarios that target IoT systems is the first major challenge [12]. The emergence of IoT-based Botnets which often perform intricate attacks against other network-enabled devices is a major threat. Thus there is a need for digital forensics for investigating Botnets in IoT.

- **Sub-problem 2**: collecting a realistic IoT network dataset which will be used to train and evaluate machine learning models is a major challenge [31]. The dataset needs to capture the normal behaviour of devices, both PCs and IoT devices connected to current networks. Furthermore, it needs to include representative botnet activities, so that the trained model can capture the patterns generated by bots in the real world.
- **Sub-problem 3**: investigating and analysing network attacks by leveraging Machine Learning [32, 33]. Each machine learning model is best suited to dealing with certain classification problems and data. The selected model needs to be able to process large quantities of data as fast as possible, without sacrificing its accuracy. Additionally, the hyperparameters of the model have to be selected carefully in order to maximize its accuracy, while avoiding overfitting.
- **Sub-problem 4**: defining attack vectors and tracing origins of hacking [34, 35]. The framework needs to cover multiple stages of the digital forensic process, including collection, preservation, examination, and analysis.

Based on the above sub-problems and the technical background, which is given in Chapter 2, the following research questions are formulated.

1. How could Botnet activities in an IoT environment be investigated effectively?
2. How could a network forensic framework for IoT networks be designed?
3. How could we handle larger volumes of data/make the classification process more efficient?
4. How could a network dataset be generated, in order to develop an investigation framework?
5. Which ML classifier type is best suited for the task of creating an investigation framework?

1.4. Digital Forensics and Machine Learning methods

The attacks that were discussed in Section 1.2, are often difficult to detect, due to obfuscating measures that botnets are often equipped with. Although the field of digital forensics has been in the focus of the research community, with many solutions developed to detect and analyze attacks in the past, a lot of them are not suited for application in an IoT setting. As such, due to the volume, the variety and velocity of network data that the IoT produces, it is evident that new solutions are needed as discussed in Chapter 2.

When investigating cybercrimes, a forensic expert needs to identify the correct tools and methods, that best suit their circumstances. As a result, digital forensic investigation frameworks were created, to provide guidelines for an investigation and ensure that identified evidence and produced deductions hold ground in a court of law. Many investigation frameworks have been proposed for IoT scenarios like the FIF-IoT, Probe-IoT and the Block4Forensic [17, 18, 19, 20]. However, they all emphasize the acquisition phase of an investigation, focusing on the distributed blockchain for preservation, and neglect to cover the examination and analysis phases, through which evidence is identified. It is thus evident that a new forensic framework is needed, in order to cover this gap.

With cybercrime increasing in frequency and sophistication over the years, digital forensics evolved by spawning several sub-fields, each specializing in locating evidence in different subsystems or file types of a device [16]. Among the various digital forensic subcategories, memory forensics deals with locating traces in the RAM, disk forensics focuses on the hard drive, malware forensics specializes in analyzing the behaviour of captured malware binaries. In the IoT era, three categories of digital forensics have been considered the most, Cloud, IoT and Network Forensics [36, 37, 38]. Cloud forensics is applied to the backend infrastructure of a cloud provider, facing many challenges including mixed jurisdiction and lack of borders between the virtual machines. IoT forensics is relatively new and deals with the extraction of data from the hardware of IoT systems, facing challenges due to heterogeneity. Network forensics investigates logs and traffic generated by network-connected devices, making it ideal for identifying a variety of network-based attacks that otherwise leave no trace. In this PhD thesis, we focus on network forensics, as it can be used to identify botnet activities without considering the hardware of the device, as most IoT systems utilize the TCP/IP protocol suite to exchange information with the cloud backend.

1.4.1. Network Forensics

Network forensics by design deals with volatile data, which is rapidly generated and transmitted, sometimes being stored in the form of logs or fleetingly existing in the RAM [38]. It can be separated into two categories with regards to its intended purpose, that of security or criminal investigation. Network forensics for security involves methods for monitoring proactively a network for anomalous behaviour. One such method often used is Network-based Intrusion Detection Systems (NIDS) [39], which function by monitoring traffic in strategic areas of the network and either matching traffic to known attack patterns or using a normal user profile and detecting any abnormalities. The NIDS can then raise alarms that warn administrators of an existing threat in their network.

Network forensics for law enforcement needs to adhere to certain rules regarding its methods and data collection. First, its methods need to be transparent and their results easily reproducible. Additionally, data collection must be conducted with care and a chain of custody needs to be established in order to ensure that the data is not contaminated [40, 41]. By doing so, the results of a network forensic investigation are deemed admissible in court. Unlike network forensics' methods for security [42], its methods for law enforcement focus on attack identification and attribution, which involves activities such as file reconstruction from captured traffic or attack detection and identification of its origins, methods used, motive and goal of the attacker. The work presented in this thesis is intended to be used for law enforcement investigations, with more details given in Chapter 5.

Different types of network forensic techniques that can be applied in different scenarios have been developed, each with their own merits and demerits. Although further discussed in Chapter 2, some of these techniques include honeypots, deep packet inspection, and network flows [43, 44, 45]. Honeypots function by imitating a legitimate device, often with limited security, in order for an attacker to target it, instead of other legitimate devices, gathering information about the attacker's activities at the same time. To design network forensics techniques, deep packet inspection (DPI) is a technique that processes both a packet's header and its body, in order to identify malicious patterns in the packet. Network flow, similarly to DPI, relies on collected traffic for its activities, although where DPI scans the entire packet, network flow relies on metadata and statistical information created by grouping packets into flows based on the source/destination IP/port, the communication protocol in use and the time [46]. In this thesis we utilize network flow because it disregards the payload information which is often encrypted, it circumvents privacy concerns and produces quicker results.

1.4.2. Machine Learning

Machine Learning (ML) is a collection of algorithms that learn to automatically identify patterns by learning from data through the use of mathematical processes [47]. Based on the method of training, ML algorithms can be grouped into unsupervised and supervised. Unsupervised algorithms are used on unlabeled data, which means the output of the algorithm is not known [47, 48]. There are many types of unsupervised learning, with one prominent example being clustering. Often, a clustering algorithm will select a number of arbitrary starting points in the data, called centres, and by using a distance metric group all the data into clusters, repeating the process until the centres and clusters do not change [49]. Other unsupervised algorithms like the Autoencoder, attempt to reproduce the input data, in order to produce a feature-reduced representation of the initial data. Many unsupervised techniques have been proposed, with some examples being K-Means, Autoencoders and Generative Adversarial Learning [48].

Unlike their unsupervised counterparts, supervised algorithms are trained on labelled data which means the expected output for the provided examples are already known [47, 50]. A supervised algorithm will then process the data multiple times, producing a value that is compared with the known output, their relative distance is measured and the resulting error is then used to alter the algorithms internal state until the predictions are similar to or identical to the known labels. Supervised algorithms themselves are further categorised in classification and regression algorithms, with the former producing an output from a limited set of values, while the latter used to predict numerical values from a continuous set. Many supervised algorithms have been developed, with some prominent examples being Naïve Bayes, Support Vector Machines, K-Nearest Neighbor, Decision Trees and Artificial Neural Networks (ANN) [50]. In this thesis we utilize Deep Learning [51], a subgroup of Neural Networks, to create a discriminative model that detects attacks.

Deep Learning (DL) is a subcategory of ANN that is separated from so-called "shallow" ANNs, by having multiple hidden units and layers [52]. Because it displayed better performance than other ML techniques when processing large volumes of data, DL has been in the ceontre of research for the last few decades. Thus, multiple versions of Deep Neural Network (DNN) have been developed, such as Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), Multilayer Perceptron (MLP). An RNN is a type of ANN, where its hidden layer in time *t* is connected to its hidden layer in time *t+1*. RNNs are best suited for data that maintain some temporal relationship between records, such as text or music [53]. CNNs are ANNs that are fully-connected and are often used to process images. Initial layers of the network capture low-level features of an image, with every layer, added increasing the complexity of the detected features [54]. MLPs are feed-forward ANNs the simplest version of deep neural networks with its units using a non-linear activation function, such as ReLU [52]. In this thesis, we chose MLP as the type of DL model to use for the creation of the Particle Deep Framework.

In order to train and test the deep model of the proposed Particle Deep Framework, a dataset with representative network traffic of a contemporary IoT network was needed. Several network traffic datasets are available for use such as KDD99, UNIBS, UNSW-NB15 [55, 56, 57]. Existing network traffic datasets had issues. The KDD99 dataset was generated to evaluate IDS and was derived from MIT Lincoln Lab's DARPA 98 dataset. The UNIBS dataset was produced through 20 workstations and incorporates normal and DoS traffic. The UNSW-NB15 dataset was developed at the UNSW Canberra, by using IXIA perfect storm which generated both normal and attack traffic, with the attack traffic covering a diverse range of attack activities. More datasets have been generated and are further described in Chapter 3. However, existing datasets displayed one or more drawbacks. Unlike the previous datasets, we developed Bot-IoT which is made up of up-to-date network traffic both normal and attacks, incorporates IoT network flows and includes new features [57, 58, 59, 60].

1.5. Thesis contributions

This PhD contributes to the field of Network Forensics, provides solutions to the above sub-problems through the following points which are key to the creation of the Particle Deep Framework.

- **Creating a new virtual testbed for the development of new IoT network dataset Bot-IoT (Chapters 3 and 4)** - Several virtual machines were interconnected in a LAN to represent a realistic real-world network. The Node-red tool [61] is used to simulate IoT devices and generate traffic, while normal traffic is produced through VM interaction and the Ostinato tool [62]. Attacks are generated through the use of Metasploit [63], Hping3 [64] and other tools that are further discussed in Chapter 3. The Tshark tool [65] was used to capture approximately 69 GB of raw network traffic, grouped in 1 GB files for portability and ease of use. The Bot-IoT dataset includes 16.7 GB and contains 72,000,000 records, while a training/testing pare was extracted, having a size of 0.78 GB (approximately 5% of the original and 3,000,000 records), to facilitate its portability and use.

- **Analysing the Bot-IoT dataset (Chapter 3,4)** - For the analysis of Bot-IoT, two high-quality datasets were chosen, the UNSW-NB15 and KDD99. Seven machine learning models are selected for the analysis against the two selected datasets, while additionally three deep learning models (SVM, RNN, LSTM) were utilized to evaluate the quality of the dataset separately. The complete machine learning analysis is provided in Chapter 4.
- **Determining the optimal features for use in defining the machine learning model of the Particle Deep Framework (Chapter 3)** - Both the Pearson Correlation Coefficient (PCC) [66] and Shannon Joint Entropy (SJE) [67] are used for the statistical analysis and determining the optimal features of the Bot-IoT dataset to be used for training machine learning models. Initially, both PCC and SJE average scores are calculated for each of the pre-existing features. These averages are then compared, and the 10 least-similar features are selected. Next, the process is repeated, with both the previously selected 10 features and the 14 new features (Chapter 3), finally acquiring the final 10 best features.
- **Developing the Particle Deep Framework, an effective Network Forensic Framework based on deep learning for Botnet related Incidence in an IoT environment (Chapter 5)** - The PDF incorporates activities that cover the stages of a digital forensic investigation process. Acquisition is handled by established software like Tcpdump or Wireshark, while Preservation is ensured through cryptographic hashing. For the Examination and analysis PDF combined Particle Swarm Optimization with Deep Learning, in order to determine hyperparameters and train a Multi-Layer Perceptron that displays significant Accuracy, Precision and Recall within a reasonable time.

1.6. Thesis structure

The thesis is organised as described in Figure 1.1.

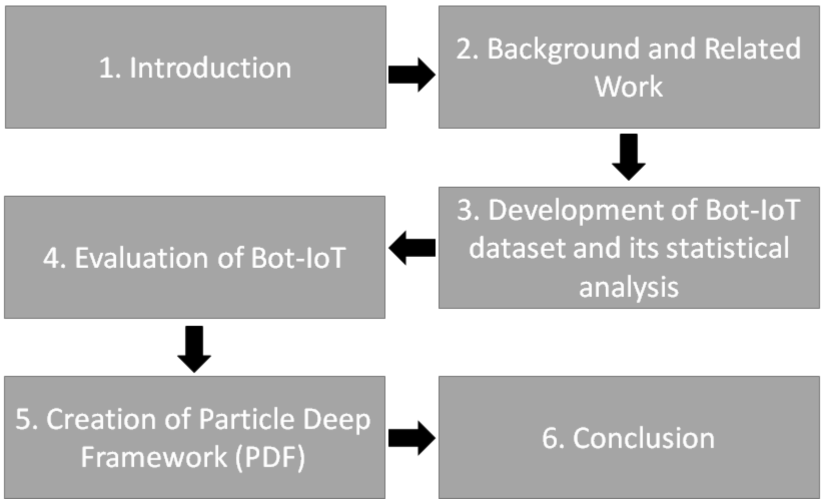

1. Introduction
2. Background and Related Work
3. Development of Bot-IoT dataset and its statistical analysis
4. Evaluation of Bot-IoT
5. Creation of Particle Deep Framework (PDF)
6. Conclusion

Fig. 1.1: Graphical outline of this Thesis

Chapter 2 provides background and related studies about the IoT, botnets and network forensics accompanied by literature related to the research of this thesis. Initially, we establish the need for our work by comparing its content with other review papers, indicating their gaps. Next, after the background information on the three aforementioned areas, the research focuses on the major network forensic methods used to investigate botnets, in both IoT and non-IoT networks. Finally, challenges are identified with regards to building network forensic solutions for investigating IoT-based botnets, brought about due to the architecture of IoT systems.

Chapter 3 explains the creation process of the novel dataset which includes IoT-generated traffic, the Bot-IoT dataset. At first, by reviewing publicly available network attack datasets, it was established that there was a lack of datasets that incorporated IoT-related traffic. That was the motivation behind creating the Bot-IoT dataset. Next, the process of creating the virtual testbed is explained, followed by network flow extraction. The rest of the chapter provides the methodology that was used for pre-processing of the data, followed by the creation of new features and the statistical process for feature reduction. The botnet scenarios are fully described and the statistics of the attacks are provided. Finally, a primary analysis of the dataset is provided, by training three different machine learning models.

Chapter 4 presents the machine learning evaluation of the Bot-IoT dataset. First, seven commonly used machine learning models are selected for the evaluation. The two datasets UNSW-NB15 and KDD99 are used to obtain initial metrics.

Next, the same types of models were trained on the Bot-IoT dataset. Obtained metrics are then analysed to evaluate the Bot-IoT dataset, and the results and their discussion are provided at the end of the chapter.

Chapter 5 details the creation of Particle Deep Framework (PDF), a new network forensic framework for IoT networks based on machine learning. At first, existing digital forensic frameworks for IoT scenarios are analyzed, proving that most focus on the acquisition stage rather than the analysis. Thus, with this gap as the main motivation, next the structure of PDF is given. The use of Particle Swarm Optimization (PSO) as a method for determining the hyperparameters of a machine learning model is explained and the deployment of the PDF in an IoT network is discussed. Finally, experimental results, depicting the performance of the PDF is provided.

The concluding remarks are given in Chapter 6, followed by Appendix A which dictates the protocols present in the Bot-IoT dataset. Descriptions of the features in UNSW-NB15, the dataset used to evaluate the PDF are given in Appendix B.

Chapter 2

Literature Review (Background and Related Work)

2.1. Introduction

The Internet of Things (IoT) has exhibited a dramatic growth over the years. With Gartner reporting that the number of deployed IoT devices around the world are expected to reach about 20.4 billion in 2020 [7], displaying an increase of 145% from 2017, it is becoming evident that this new diverse domain will continue to grow as companies discover the benefits of IoT services.[1]

As these numbers are on the rise, a growing concern for IoT systems is their security and privacy. In a study by Hewlett Packard in 2015 [10], it was shown that out of a number of IoT devices that were investigated, 80% raised privacy concerns, with 60% lacking any mechanisms that verify the authenticity of security updates or even their integrity, allowing an adversary to modify the firmware without being noticed. Another example of IoT vulnerability is the study by Ling et al. [11] with its focus being a smart plug, a device that provides automation to mundane electronic equipment (fans, heaters). They were able to compromise the device and perform a number of attacks, one of which was a firmware attack, where an attacker can modify the device's firmware, gaining the ability to install malicious code to the device.

Seeing as IoT devices are manufactured with various pre-existing inherent limitations and vulnerabilities, it should come as no surprise that they have been targeted and recruited by botnets. Having the advantage of being designed to function 24/7, botmasters lately have shown their preference for using these devices instead of the better protected, not so reliable PCs and Laptops [68]. In their quarterly report on the state on the Internet security for the 4th quarter of 2016, Akamai highlighted that we experienced the third wave of botnets, with the

[1]The work presented in this chapter has been published in:
N. Koroniotis, N. Moustafa and E. Sitnikova, "Forensics and Deep Learning Mechanisms for Botnets in Internet of Things: A Survey of Challenges and Solutions," in IEEE Access, vol. 7, pp. 61764-61785, 2019. doi: 10.1109/ACCESS.2019.2916717

emergence of IoT-based botnets, with the first two harnessing PCs and servers respectively as bots [69]. One such prominent example of this new wave of botnets is the Mirai botnet. It was first observed performing DDoS attacks against journalist Brian Krebs' blog, with the first DDoS peaking at 623 Gbps (77.9 GBps) and latter attacks targeting French web-host and cloud service provider OVH reaching 1.1 Tbps [68, 69].

Such attacks, apart from a discrediting factor that can affect the ability of a company to be perceived as trustworthy and reliable, can also have more immediate monetary repercussions, as most companies targeted by DDoS attacks rely greatly on their Internet connection to provide their services. Alternatively, some botnets have been designed to launch several types of diverse cyber-attacks such as identity/data theft (data exfiltration), where the Bot-code infecting a machine gathers sensitive user information and sends it to the botmaster, e-mail spamming, where infected machines are used to produce and send fake e-mail, keylogging, where the user's input is logged and transferred to the botmaster and malware propagation, through which a bot is used to further propagate a malware to its network neighbours and/or other Internet nodes [22, 70]. With such destructive attacks on the rise, it is clear that security and forensics in the IoT should become a priority for research.

The main contributions of this chapter are as follows:

- We provide a comprehensive background for the IoT, botnets and forensics.
- We provide a taxonomy of recent methods for botnet identification and tracking.
- We provide a new definition for the IoT, which focuses more on the "Things".
- We investigate the applicability of deep learning in network forensics and the inherent challenges that appear when network forensics techniques are applied to the IoT.
- We determine future directions for research related to performing forensic investigations of IoT powered botnets.

Chapter 2 is structured as follows. Background information for IoT, Botnets and Digital forensics is provided in Sections 2, 3 and 4 respectively. Furthermore, in Section 4 a number of forensic solutions are listed and grouped into categories based on the technique. Section 5 presents background information about deep learning followed by network forensic applications based on deep learning. Finally, challenges and future trends are presented.

2.2. Internet of Things (IoT)

This section explains IoT concepts, growth of the IoT and its areas of application, as well as IoT models and systems.

2.2.1. IoT concepts and definitions

Over the years, the IoT has evolved in complexity and functionality, maturing and becoming an integral part of society, spanning multiple fields of application. The concept of IoT has existed for quite a while, sometimes under a different name, like 'ubiquitous computing', 'embedded intelligence', 'web of things', 'Internet of objects' and 'ambient intelligence' [3]. The term 'Internet of Things' was introduced by Kevin Ashton during a presentation in 1999 [71], where he explained the value of having computers that gather and utilize data in an automated and contextual fashion. In literature, IoT has various definitions, which we summarize in Table 2.1, along with their individual advantages and disadvantages.

Interestingly, the first time the “essence” of IoT was embodied, was around the beginning of the 1980s at the Carnegie Mellon University where a soft drink dispenser was coded to allow users to remotely view the availability of certain drinks [3], followed by Cambridge's Trojan coffee room, where a similar logic was applied to a camera which was used to check the amount of coffee remaining in a pot [75]. In 2000, Ashton et al. [76] produced a white paper depicting their views of the new MIT Auto-ID Center, where they described a world filled with objects connected to one another and tagged with relevant information, a vision similar to the RFID technology. From then onward, a number of events occurred which shaped the IoT into its current form. The first major adoption of this new idea was in 2000 when LG announced their plans to launch the first 'smart' refrigerator which could determine if the stored supplies were running low [75, 76], while a more formal introduction to the IoT was given by the International Telecommunication Union in 2005, through a report titled 'the Internet of Things' [76, 77].

Table 2.1: Overview of IoT definitions

Definition	Merits	Demerits
Ashton et al.[71]: A set of systems in which the Internet is connected to the physical devices using several sensors and actuators.	This definition shows that IoT links the physical and virtual world. It is a direction of executing tasks quickly.	The definition does not consider the scalability and compatibility of IoT's appliances.
Haller et al. [72]: The seamless interconnection of the information network and physical objects, called 'smart objects', with these object being active participants in business processes, being accessed through network services, with security and privacy in mind.	This definition shows the important role of the IoT for businesses, as a source of services to be provided. It also suggests that security and privacy need to be considered when these devices are accessed.	This definition provides a business point of view, excluding home IoT systems. It does not consider the security of individual IoT device or the system as a whole. Scalability issues also not mentioned.
CERP-IoT report [73]: A dynamic global network, capable of self-configuration, based on interoperable standards and protocols, with 'things' having both physical and virtual aspects, built in a way to be seamlessly interoperable. Things are able to interact with themselves and their surroundings and can be manipulated securely by users.	This definition acknowledges the duality of IoT devices (physical and virtual 'things'), mentioning also the concepts of interoperability and self-organization.	The definition does not explain how the IoT will be a self-sustainable system.
Gubbi et al. [74]: A cross-platform interconnection of sensors and actuators, capable of sharing information, and achieved through a combination of ubiquitous sensing, data analytics, visualization techniques and Cloud as a unifying framework.	This definition describes the interoperability and interconnection, which make up the IoT, emphasizing more on the technologies that can be used for implementation.	Asserts that the unifying framework, should be Cloud computing, other technologies are not considered. Scalability and security are not considered. IoT enabled services, not in the scope of the definition.
Madakam et al. [3]: A network of intelligent, self-organizing objects, capable of sharing information and resources, sensing the environment and reacting to changes.	This definition shows that the IoT should be scalable and that IoT devices should be able to react to sensory input.	The definition does not describe any user interaction with the devices, also security is not considered.

In 2008, IPSO alliance was formed to promote the adoption of the Internet Protocol (IP) for the communication of “things”, in what appeared to be the first step to start setting up common practices among the many vendors. Although work had been underway to develop the IoT, in one way, it was the creation of IPv6 that truly enabled its rapid development, as it allowed for a virtually unlimited number of devices to be connected [75, 76]. Finally, in 2014 the Open Interconnection Consortium was promoted as an open framework for the Internet of Things by Intel and other firms [75]. Even so, there is still no standard framework for the IoT commonly adopted in industry, which forces vendors to decide on their own how to implement their devices, hindering somewhat the interoperability between differing IoT implementation.

The current diverse technologies involved, the fact that multiple communication protocols could be in use in a single infrastructure, and the mobility that characterizes the IoT, make it polymorphic in nature and contribute to the difficulties of pinpointing a single definition that best describes it in its entirety. Consequently, we provide the following comprehensive definition for the IoT:

Definition. *“The IoT is a network of networks comprised of devices, small and large named ’things’, that have been imbued with finite amounts of processing power and communication capabilities providing services, including software, platforms and infrastructures to a remote user/organization on-demand, with lower cost than purchasing physical systems ”.*

In other words, IoT is the creation of networks where machine-to-machine communication is used between geographical locations, industry/business sectors and other entities, whereby there is no direct communication. This can either enable software applications through the sharing of data or allow for the direct intervention of the environment where these IoT devices have been deployed. Such devices could be as complex as smartphones, which have multiple sensors and significant processing power or as simple as smart lightbulbs, that enable control of lighting conditions in a large environment such as universities [26].

2.2.2. Growth of IoT and its areas of application

Promising innovation, automation and optimization of industrial and commercial systems, no one should be surprised by the worldwide growth that the IoT market has experienced, with multiple studies and predictions made for it. A study by Gartner for instance calculated that in 2017 IoT deployed devices will reach 8.3 billion and project that the numbers will skyrocket to 20,4 billion in 2020 [7]. The predictions made for the economic growth of the IoT are better understood if one considers the fact that the IoT does not cater to only a single portion of the world market, but instead slowly becomes an integral part for most fields in today's society.

In a white paper by the Internet of Things Alliance Australia (IoTAA), a segmentation of the IoT field into domains each of which describes a specific market was provided, with the domains being: Consumer, Industrial, Healthcare, Smart City, Automation, Agriculture, Critical Infrastructure [6]. Applications of the IoT for these domains appear constantly and in inventive ways. In the Agriculture domain, sensors for monitoring environmental data, such as levels of moisture in crops, airspeed and temperature, placed the underpinnings for a future system [78]. In the Automotive domain, a monitoring system transmits and displays location and diagnostic data through a Cloud provider, improving the driving experience and assisting in determining the optimal time for mechanical services [79]. Finally, in the Healthcare domain, the incorporation of IoT devices has been shown to benefit patients, as vital information can be gathered from the comfort of their home, contributing thus to detecting quickly any deterioration in their condition [80].

A. IoT models

As previously mentioned, the IoT has no single commonly accepted standard framework or set of standards. Instead, vendors are free to implement their systems by using the technologies they prefer, resulting in a heterogeneous IoT environment. The IoT is by design vast, spanning multiple technologies, which need to coexist in perfect harmony for the whole system to be functional. There are multiple IoT designs and models.

First of all, from a communications point of view, the Internet Architecture Board (IAB) in a guiding architectural document released in 2015 described four communication IoT models [8]. First, the Device-to-Device Communication model, where devices communicate directly with one another. Such type of communication is primarily present in home automation IoT and usually relies upon Bluetooth, Z-wave or ZigBee as the communication protocol, as they are ideal for the exchange of small amounts of information in relatively small areas. A drawback of this model is that it requires its devices to collectively use the same communication protocol, limiting the devices that can be employed in this configuration.

In contrast, a more versatile model is the Device-to-Cloud model, where devices connect directly to a Cloud service provider to store collected data or receive instructions. This type of model allows the end-user to access their device through a Web interface or a smartphone app and view reports from a data collection, or change the state of the device. This model's drawback is that in most cases, the Cloud provider and the Vendor who produced the device are one and the same, denying users from using a Cloud service provider of their choosing, a situation that is called 'vendor lock-in'.

Thirdly, an evolution of the Device-to-Cloud model is the Back-end Data-Sharing model, which is an exact duplicate of the Device-to-Cloud model, with the added bonus that the user can extract data from the original Cloud provider and transmit it to other Cloud providers. This allows for data aggregation and has the benefit of giving the user the freedom of moving his/her data between Cloud providers.

Finally, in the Device-to-Gateway model, a device connects to the Cloud service provider through an Application Layer Gateway service, running on a local machine which functions as a proxy. The gateway in this model, apart from providing secure connectivity to the Cloud, allows for devices which use different communication protocols to interact, enhancing interoperability. In real-world scenarios, in some situations, smartphones play the role of the gateway, with examples such as fitness tracking devices. In other scenarios, a 'hub' is used, which is a dedicated device that plays the role of the gateway and is most commonly found in the home automation scene.

An alternate way to view the IoT is given by the four-typed model, which splits the IoT into four layers [81]. The sensing layer, consisting of sensors and actuators, which enable perception of the world and the ability to act through the IoT. The networking layer, which handles communications between various network systems including heterogeneous devices of the IoT. The service layer, which allows applications to connect smoothly to the services provided by the IoT through the use of middleware. And the interface layer, which provides a means of interaction between various services in a system with the front end application.

The IoT ecosystem, which included its various components and the way with which they interact, can be viewed as follows, and as shown in Figure 2.1 [1].

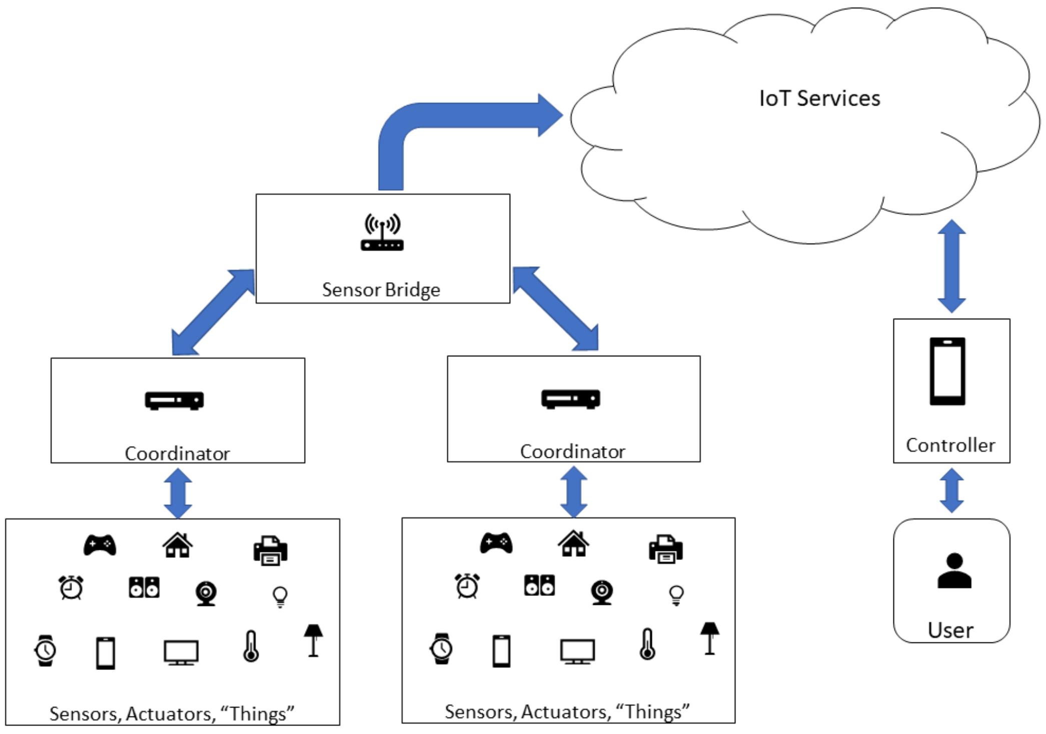


Fig. 2.1: IoT Ecosystem adapted from [1]

First, the IoT devices, sensors and actuators collect information and perform actions. The devices then, through Coordinators, connect to the local Sensor Bridge which functions as a gateway, enabling at the same time interoperability between different protocols and technologies. Coordinators are tasked with health monitoring, data forwarding between devices and service provider and the creation of reports about all actions taken, while the Sensor Bridge connects the various heterogeneous IoT sub-networks with the service provider in the Cloud. The IoT Service handles many tasks, some of which are data storage, data processing and device management. Finally, through a Controller, the end-user is able to connect the IoT Service and through that manage their devices.

2.2.3. IoT technologies

As previously mentioned, the IoT is an amalgamation of several technologies and protocols employed on different levels of its ecosystem, enabling its functionality [3]. Different IoT technologies have emerged in the industry, for example, Radio

Frequency Identification (RFID), is used broadly as a cheap identification method for devices. For global communication between gateways and cloud service, the Internet Protocol (IP) is preferred, where both IPv4 and IPv6 are in use, with the latter allowing for close to 85,000 trillion IP addresses. For local communication between IoT devices and their coordinators, the most prominent technologies employed are Wi-Fi, Bluetooth, ZigBee, Z-Wave. Finally, regarding the actuators, they are generally split into three categories: Electrical, Pneumatic and Hydraulic, based on the medium they use for power. These technologies are vulnerable to cyber-attacks due to the IoT open-loop of communication, and heterogeneity of their protocols and services. We mainly focus on botnets, as they constitute considerable harm for IoT appliances and applications, as explained in the following section.

2.3. Botnets

This section provides information related to Botnets, their origin, architecture and activities.

2.3.1. Botnet background

Botnets have had a rich history and development over the years, corrupting and disrupting computer and network systems [82]. Originally, botnets were crafted for benevolent purposes, with their main function being to provide administrative assistance to Internet Relay Chats (IRC), a form of communication quite popular in the '90s. The first IRC bot appeared in 1993, was named Eggdrop and provided assistance to IRC channel communication. Following Eggdrop, the first malicious bots made their appearance, with GTbot in 1998 being the first of its kind, which was able to execute scripts when prompted through its Command and Control (C&C) IRC channel.

In 2003, new more sophisticated bots appeared like the Agobot, which was more robust and flexible than previews types, as-well-as it incorporated a persistent C&C channel. In 2007, Storm made its appearance, a botnet that was characterized as one of the most powerful botnets of its time. It employed a P2P C&C infrastructure, with its main functionality being spam messages, DDoS attacks and had the capability to disrupt the Internet communication flow of entire countries. In the same year, appeared probably one of the most infamous botnets, the Zbot or Zeus. Having at the time close to 3.6 million bots under its control, other variants were later spawned, including a P2P version in 2011 named Gameover Zeus, which was capable of performing a wide range of malicious activities, including bank account theft, DDoS and spam [83]. It was eventually taken down by a collaboration of the FBI, the UK NCA, Shadowserver Foundation and Dell's CTU IN in June 2014 [84].

In 2008, some notable botnets that appeared were Asprox, Kraken, Torpig and Conficker. Asprox, like the original Zeus, used a centralized HTTP based C&C infrastructure and apart from its main purpose, which was the creation of spam, it was also capable of performing SQL Injection attacks to websites. Kraken was part of the spammer botnet family and was reported that in April 2008 included 400,000 bots in its army of zombies [85]. Torpig was a data exfiltration botnet, which performed man-in-the-browser attacks and used a centralized HTTP-based C&C infrastructure [86]. Conficker moved periodically from Centralized HTTP-based C&C to P2P [87].

The successor of the Storm botnet, Waldec was discovered in 2009, having a P2P C&C infrastructure, its primary function was to send spam messages reaching close to 7000 messages per day, though it also performed credential theft and DDoS attacks [87]. Eventually, it was taken down in 2010. Also in 2009, Mariposa was discovered. Mariposa used a custom communication protocol which was a variation of UDP, was capable of launching DDoS attacks and even download and run executables, such as other bots. It was taken down in December 2009 [88]. Another notable milestone for botnets in 2009 was the appearance of the precursor of mobile botnets, where botnets use mobile phones as their bots (zombies), named SymbOS\Yxes which targeted Symbian devices and utilized SMS messages to self-propagate [89, 90].

Following the surfacing of SymbOS, the first botnet targeting Android devices named Geinimi was observed, during the end of 2010. Primarily found in China, it employed a simple HTTP-based C&C infrastructure and was capable of sending SMS, e-mails, fetch the location of the infected device and also made possible the further propagation of malware [89, 91]. Lately, botnet creators have taken advantage of the wide adoption and constant increase of the IoT, and we have already seen examples of IoT botnets and what they are capable of.

Botnets comprised of IoT devices were the next evolutionary step of botnets. The most well known first appeared in September 2016, under was aliased as

Mirai [68]. Mirai performed some of the most powerful DDoS attacks in Internet History, namely: 620 Gbps against Brian Kreb's website, 1.1 Tbps against French Cloud service provider OVH and in October 2016 attacked Dyn service provider and took down portions of the internet like Twitter, Netflix and GitHub. After the release of Mirai's source code, various variants appeared like Persirai which is active since April 2017, a more refined version of Mirai which targets specific devices of select vendors. Other IoT botnets include Hajime, which appeared in October 2016 and utilized a decentralized C&C infrastructure which appeared to 'shield' devices from Mirai infections. Finally, BrickerBot was observed in April 2017, and as the name suggests attempted to 'brick' IoT devices in what can be considered a permanent DoS attack.

2.3.2. Botnet architectures and characteristics

Botnet architectures include several elements. To start with, a bot is a program which, after reaching a vulnerable host, infects it and makes it a part of the Botnet [82, 92]. Bots differ from other malware, in that they include a channel of communication with their creators, allowing them to issue commands to their network of bots (i.e., zombies) and thus making botnets versatile when it comes to their functionality [82, 92]. A botnet's malware gets delivered to vulnerable targets through what is known as a propagation mechanism. Most commonly there exist two types of propagation, passive and active.

Passive propagation techniques require users to access sites, emails or other compromised network elements and through user interaction download the malware (bot), infecting it and making it part of the botnet [70, 92]. Active or self-propagation techniques employ sub-portions of their network to actively scan the Internet for vulnerable devices, attempting to exploit the identified vulnerabilities, turning the compromised hosts into bots themselves [70, 92].

The characteristic that makes botnets unique is the fact that they allow their controller, commonly referred to as a botmaster (a.k.a bot-herder) to issue instructions to their network of infected devices and receive feedback, as shown in Figure 2.2. This is made possible through a Command and Control (C&C) infrastructure. There exist multiple different types of C&C infrastructures based on their topology and those types are: centralized, P2P, hierarchical and hybrid [70, 93]. In a centralized topology, bots connect, receive instructions and report/deliver their work in a central infrastructure, with most common technologies employed here being IRC and HTTP protocols [82, 93]. The main drawback of the centralized topology is that the C&C is a single point of failure.

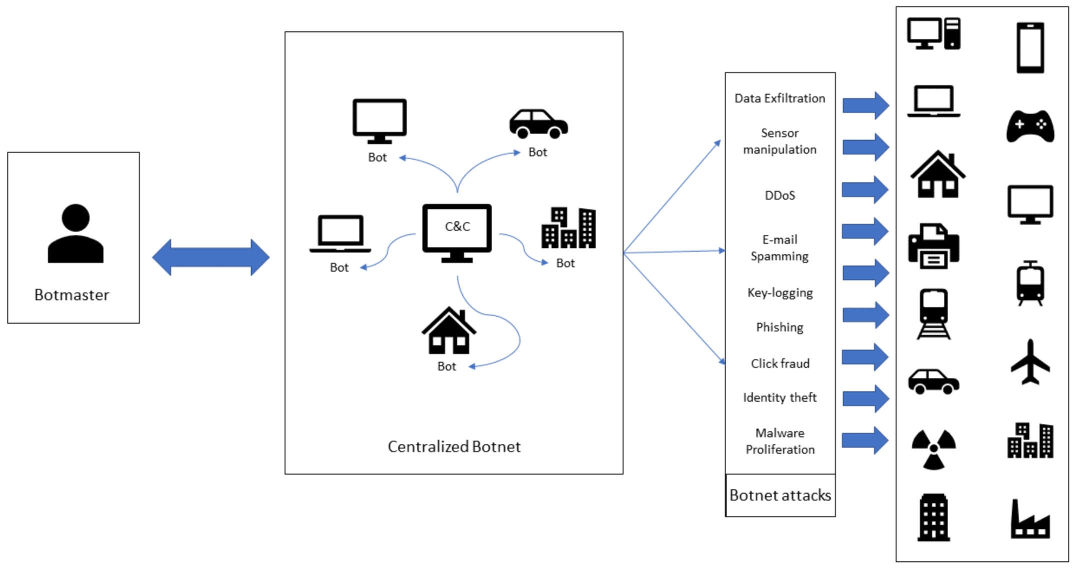


Fig. 2.2: Centralized Botnet and activities [2]

A decentralized or P2P architecture is the natural successor of the Centralized, where the bots can assume either the responsibilities of a C&C server or a worker bot that performs tasks on behalf of the botmaster. Such architectures generally face higher latencies than preferred regarding command distribution, though they are quite resilient to takedown attempts, as the compromise of a single host would only affect a small portion of the botnet [92, 93].

Hybrid architecture is, in a way, a combination of the P2P architecture with the Centralized, reaping the benefits of both [92, 93]. Here, the C&C is implemented in P2P form, with the bots that make up the C&C (called servant bots) forwarding commands to each other and to the bots that perform the actions (client bots). Finally, in this architecture, a botmaster adds proxy bots between their machine and the botnet, with each bot forwarding commands to the bots that they compromised, creating a hierarchical topology and making takedown attempts difficult, as-well-as allowing the botmaster to rent portions of their botnet.

2.3.3. Botnet activities

Botnets are some of the most versatile pieces of code to traverse the Internet.

The main reason why they get so much attention is not because of the masterful ways that botmasters employ to obfuscate their bots from law enforcement, but rather the practical capabilities that botnets possess and the services they provide to the botmasters and their clients. There are various hacking techniques used by botnets, including Distributed Denial of Service attacks (DDoS), Keylogging, Phishing, Spamming, Click fraud, Identity theft and even the proliferation of other Bot malware [22].

Botnets tend to be specialized into performing a small subset of the aforementioned hacking techniques, though there have been cases where variants of botnets were capable of multiple types of malicious activities, an example being Gameover Zeus which could perform DDoS attacks, send spam e-mails and steal bank account information. There are many ways a botnet can perform a DDoS attack, and based on the technique and protocols employed, there are multiple examples of such attacks, some of which are: Denial of Sleep attack, UDP flood attack, TCP SYN flood, ICMP ping flood, Ping of Death, Smurf attack, DNS amplification, HTTP flood [94, 95]. In a Denial of Sleep attack, the attacker targets functionality provided by the Medium Access Control (MAC) layer, where devices are set into a 'low power mode', to decrease battery consumption, which is of vital importance for network sensors [94].

A UDP flood attack exploits the connectionless nature of the UDP protocol and sends a large number of forged packets to random ports of the target machine, forcing it to expend resources to detect any applications which could be waiting to receive the incoming information and then issuing ICMP responses when that check fails [95]. On the other hand, TCP SYN flood attacks utilize the structure of the 'three-way handshakes' that is performed in order to set up the parameters for any TCP connection. In this case, the attacker floods the target with SYN packets, which are used to initiate a TCP session, receives the response from the target but does not send the final packet that establishes the connection, forcing the target to maintain the connection open and thus eventually cause the target to become unresponsive [94, 95].

An ICMP ping attack is similar in nature to the UDP attack, in the sense that a large number of packets are sent to a target which forces the target to respond, taking up network and processing resources [94]. A Ping of Death, though it utilizes ICMP is somewhat more interesting, as it exploits the fact that the IP is designed with an upper limit of 65,535 Bytes, and when a larger packet is received, it causes memory overflow issues and eventually crashes the machine [94, 95]. Finally, a Smurf attack consists of a large number of ICMP packets which have the intended target's IP address spoofed in place of the packets' source address, causing the replies received from such packets to be sent to the target [95]. When a botnet performs keylogging, it silently records the keystrokes of a user and after a certain amount of time, it sends its gathered information to the botmaster. Phishing is the process through which the adversary attempts to trick a user in revealing sensitive information, such as login credentials or even bank account information, through carefully crafted messages, websites, and emails. Finally, botnets are sometimes used to proliferate other malware, with spam email being one way to do so.

There are different security controls, for example, threat detection and intelligence, as well as intrusion detection techniques, have been used for recognizing and preventing botnets from network and IoT systems. These techniques are beneficial to some extent, because they can only detect knows cyber-attacks, but they cannot identify zero-day attacks (i.e., new/future attacks), as there are no signatures of those attacks stored in blacklists. This is the motivation of focusing on digital forensics mechanisms in order to track and define cyber-attack origins, and assist in examining how botnet structures occur in IoT systems; hence improving security controls in discovering known and new botnets.

2.4. Digital Forensics

This section provides information related to digital forensics, its origin, investigation models, sub-domains and developed methods for investigating botnets in multiple fields including the IoT.

2.4.1. Origins and evolution of digital forensics

As criminal activities moved to cyberspace, with cybercriminals exploiting systems to their own ends, it was only natural that law enforcement would also adapt their operations accordingly, as such, digital forensics was coined. Its roots can be traced back to 1984, when law enforcement entities, among which was the FBI laboratory, started developing programs in order to examine computer-related crimes [96, 97]. Over the years, many organizations have proposed their own definitions and standards for performing forensic investigations in the digital world, with multiple investigation models appearing, most of which share some common phases but are designed to be applied in different circumstances [96].

One such definition that describes the essence of digital forensics, is the one given by Rodney McKemmish, where he states that digital forensics is *“the process of identifying, preserving, analyzing and presenting digital evidence in a manner that is legally acceptable”* [15]. An investigation model proposed by the first Digital Forensics Research Workshop in 2001 named the DFRWS Investigation Model, which functioned as an inspiration for other such models, comprised of six phases: Identification, Preservation, Collection, Examination, Analysis and Presentation [97], as shown in Figure 2.3.

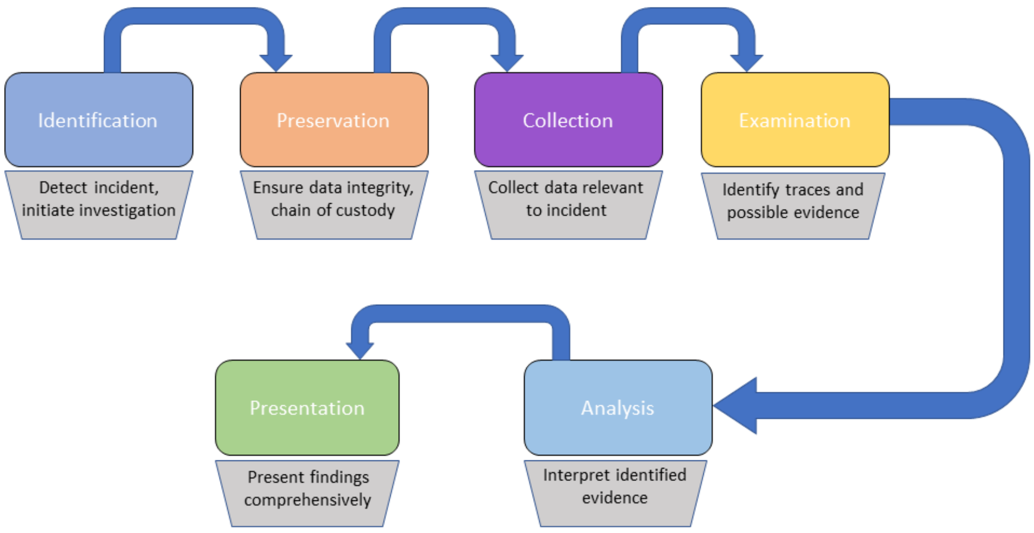

Identification
Detect incident, initiate investigation
Preservation
Ensure data integrity, chain of custody
Collection
Collect data relevant to incident
Examination
Identify traces and possible evidence
Presentation
Present findings comprehensively
Analysis
Interpret identified evidence

Fig. 2.3: Phases of digital forensics mechanisms (DFRWS Investigaiton Model)

In the Identification phase, sources of possible evidence are identified. This phase takes under consideration that the amount of data an investigator can collect is constrained [96]. In the Preservation phase, proper chain of custody is established, and further actions are taken to ensure the integrity of data to be collected [96]. During the Collection phase, the investigators make use of appropriate techniques and tools to safely collect the data which has been identified as important for the case.

The Examination and Analysis phases are considered to be of utmost importance, as here the collected data is scanned, filtered and processed in order to identify crucial evidence and establish timelines which are then provided in reports during the Presentation phase [97]. During the Investigation phase, a range of devices could be investigated for potential evidence, from mobile phones and laptops to routers and lately even fridges and light-bulbs. Some well-known digital forensics investigation models are listed in Table 2.2.

Over the years, digital forensics has been further partitioned into subfields, each of which provides specialized techniques for investigating security incidents in different domains of the IT sector. Popular forensics sub-fields are Network forensics, Cloud forensics, and IoT forensics [16], as described in the following points.

- **Network forensics** - emerged as a way to identify, understand and ultimately amass evidence to pursue legal action for malicious activities that used the Internet and other networks as a bridge for attacks with some examples being DDoS attacks and data theft [38]. In a network, evidence is usually short-lived, as packets are produced from one device and sent through intermediary nodes to their destination. As such, various Network forensics techniques have been developed over the years [38, 98, 99]. Famous tools in network security are Intrusion Detection Systems (IDS) and Honeypots [100, 101, 102]. IDSs are trained and validated to recognize patterns of malicious traffic in the network [98], while Honeypots mimic vulnerable legitimate devices, luring hackers and botnets into attacking them, with the added bonus of allowing investigators to observe what actions are performed by the attacker [38].
- **Cloud forensics** - is a branch of digital forensics tasked with the investigation of security incidents in the Cloud [36]. It is a cross-disciplinary field combining disciplines like computer device forensics and network forensics, which poses some unique challenges, like difficulty in defining jurisdiction, as a Cloud provider could be based in Europe and be providing services in the U.S., breach of privacy, as a machine that could be investigated could host services for multiple users, including suspects and an increase in generated data quantity, as an ever-increasing number of devices utilize the Cloud [36].
- **Malware forensics** - is a discipline of forensics, which focuses on reverse engineering, and analyzing the source code of malware [103, 104] acquired from captured binaries. Analysis of malware samples can be categorized as *static, dynamic or code*, depending on the methodology used. In addition to these methods, virtual machines have also been used, as they provide a resilient environment where malware behaviour can be observed in relative safety. Over the years, attackers have started to incorporate anti-forensics

logic in their code, to elude detection [104]. With anti-forensics techniques, malware infections become more resilient, for example, allowing it to alter its behaviour if it identifies that it is running in a virtual environment.

- **IoT forensics** - is an emerging new field of forensics for investigating cybercrimes by analyzing IoT devices, protocols, in terms of software-, platforms-, and infrastructure- as services. IoT forensics is slowly being developed as in [37]. Major challenges that are hindering the adoption of conventional digital forensics techniques for investigating incidents in the IoT are the heterogeneity of systems and data, the high quantity of data produced as the number of IoT devices rises constantly and the speed with which data is generated.

A. Network Forensic methods for investigating Botnets

Investigating botnets is a multifaceted problem. It requires interdisciplinary actions to be taken, to ensure effective analysis and a more spherical understanding of an infected network's actions, enabling the design of better countermeasures, or at the very least attribution. As mentioned above, Network Forensics is the branch of Digital Forensics, where the evidence is network-related, and thus exist in the form of logs, packets and network flows. In this section, we focus on Network Forensic techniques, which have been developed to analyze Botnet activities, their general characteristics (lifespan, size) between the years 2011 and 2019. These techniques have been organized into distinct methodologies as follows:

- Honeypots
- Network Flow Analysis
- Deep Packet Analysis
- Attack Recognition
- Visualization of Network Traffic
- Intrusion Detection Systems (IDS)

Table 2.2: Well-known digital forensics investigation models

Author	Model	Description
Pollitt M. [15]	Computer Forensic Investigation Process 1984.	An investigation model general enough to be applicable in a wide range of computer-related crimes. It was proposed, as a parallelism to the conventional evidence handling process. Comprised of four steps, it does not include steps for preparation, investigation and subsequent returning of evidence to their rightful owner.
Palmer G. [105]	DFRWS Investigative Model 2001.	A general-purpose investigation model proposed during the Digital Forensics Research Workshop (DFRWS) in 2001. It somewhat expanded the Computer Forensic Investigation Process, by trading the Acquisition step for three new steps, Identification, Preservation and Collection, thus emphasizing the necessary steps to identify potential sources of evidence, secure them in a way that proveably prevents any alterations to their state and then proceeds with collection and the subsequent steps of the investigation.
Baryamureeba V. et al. [106]	Enhanced Digital Investigation Process Model (EDIP) 2004.	An extension of the previously proposed Integrated Digital Investigation Process model, the EDIP has the advantage of not only considering the digital crime scene but also the physical. The EDIP model includes: Readiness phase, Deployment phase, Traceback phase, Dynamite phase and Review phase, and each phase is comprised of multiple sub-phases. By partitioning the possible actions that an investigator needs to take, and including a traceback phase, this operational model is one that can be easily applied to real-world scenarios.

These methodologies are further discussed in the following segment.

- **Honeypots-[43, 107, 108, 109, 110, 110, 111]-** are devices, many times simulated ones which run in a controllable virtual environment, that have been designed to appear as an appealing target to attackers and malware infections. Honeypots are generally separated into high interaction and low interaction honeypots, with the former imitating entire systems (e.g., Windows, Linux) and allowing for extended interactions and information gathering, while the latter simulates certain services available through the net, which are often targeted by attackers. The benefit of this method is that the attacks, malware binaries and access attempts, are monitored and logged by the Honeypot operators, allowing for the generation of rules to predict similar future attacks. Additionally, it enables the extraction of malware binaries and communication patterns between C&C and bot, while it does not take part in any of the attacks issued by the botmaster.

A low-interaction honeypot system was proposed by Pham et al. (2011) [108], who focused on developing an automated and systematic way of identifying Botnet attacks in vast datasets. By employing the services of *Leurré.com*, which includes multiple machines in more than 25 countries, running low-interaction honeypots and collecting network traffic, they were able to show that by grouping together closely correlated traces of attacks, they were able to identify attack events. Kumar et al. (2012) [109] proposed a distributed virtual and fully automated Honeynet architecture, capable of dynamically reconfiguring itself. Their proposed system is partitioned into three components. In this system, a distributed honeynet client, which is comprised of a combination of low interaction and high interaction honeypots, would run in a VirtualBox with the incoming and outgoing traffic moderated by a Honeywall.

A method for extracting Intrusion Detection System (IDS) rules from data collected from Honeypots, was developed by Mittal et al. (2016) [107]. The researchers employed a Support Vector Machine, which they trained by using data collected from a Honeypot to produce the necessary rules for the IDS. Not much information is given regarding the source of the Honeypot-produced dataset or the parameters of the SVM that was used. Although Honeypots can produce extensive information regarding Botnet activities, they require huge amounts of storage space to maintain all the traces, extracted payloads and malware binaries. Low and High interaction honeypots have their tradeoffs, with the former needing a lot of effort to pass as a legitimate device, and the latter running the risk of being taken over completely, thus requiring data control measures to be taken, so that the honeypot does not take part in any malicious activities itself.

Attackers have been known to target social networks, where they are capable of gaining sensitive information from unsuspecting users, proliferating malware infections and more. As such, Paradise et al. (2018) [110] presented

ProfileGen, a tool that produces realistic and compelling social profiles, which function as honeypots, in that they attract the attention of attackers. The proposed system makes use of an automated process that relies on the generation of a Markov model from collected data. Emphasis was given in generating realistic education records for the crafted profiles.

It is not uncommon for cyber-attacks to span large segments of the Internet, with a prominent example being DDoS attacks. To that effect, Jeong et al. (2018) [111] worked on tracking large-scale events. Apart from improving accuracy, emphasis was given on reducing the communication toll due to sensors reporting an observed event in a distributed monitoring system, akin to the honeynet. In the proposed detection protocol *Bitmap-Based Widespread Event Detection*, bitmaps are exchanged between the agents and the coordinator and are used by the latter to identify events monitored by all agents, which are then categorized as widespread events. This approach improves on previous schemes, as it does not produce any false positives.

Naik et al. (2018) [43] proposed a fuzzy-based technique that identifies fingerprinting attacks in a low-interaction honeypot. This technique identifies abnormal TCP, UDP and ICMP traffic, and uses fuzzy logic to produce a probability of a fingerprinting attack targeting the honeypot. One shortcoming of this technique is that new fingerprinting attacks that rely on different patterns might not be identified effectively.

- **Network Flow Analysis-** [44, 112, 113, 114, 115, 116]- uses metadata gathered from network communication, to make inferences regarding the legitimacy of the traffic under scrutiny. Traffic is aggregated, and metrics are collected to create Network Flow Records. Records are identified by the source and destination IP addresses, port numbers and the protocol used for the communication. The benefit of this approach is that privacy is no concern, as the actual data being communicated isn't investigated and it doesn't require extensive storage capacities, as traffic is aggregated, with only certain metrics maintained. At the same time, it is not affected by encrypted communications, a problem faced by Deep Packet Inspection (DPI) while at the same time it does not make use of signatures to identify network attacks, as it relies on statistics and the application of machine learning.

  In their work, Francois et al. [116] created a system which focused on identifying members of P2P botnets. In their implementation, they employed

Hadoop, the open-source implementation of MapReduce, to run the PageRank algorithm on trace data collected by honeypots. Hosts with high connectivity to each other were categorized as potential P2P bots, as this characteristic was deemed a good indication that a host is part of a P2P botnet. Their approach, performed adequately on certain types of P2P botnet topologies, where the linkage between bots is high, although some legitimate P2P clients could be misrepresented as bots if no prior knowledge is used to fine-tune the PageRank process. Bijalwan et al. [115], focused on the investigation of UDP flooding attacks. In their experiments, they worked on identifying randomized UDP flooding attacks, which can be designed to pass undetected by IDS systems and other security mechanisms. They simulated an attack environment, by developing scripts which would extract the source IP address of the user that would be targeted, and then generated the randomized attack payload (forged packets).

Detecting suspicious patterns in network traffic by utilizing a Linear Regression model was the focus of Divakaran et al. [113]. Their solution relies on the combination of network flows into *sessions*, and the detection of illegitimate TCP states by using Finite State Machines to determine whether an attack is taking place, gathering evidence in the process. Oujezsky et al. [44] focused on time behavioural analysis, by extracting information from Network Flow data aggregations. The researchers employed survival analysis, a technique based on probability theory, developed to study the duration of virtually any process and focused on the identification of C&C communication, which tends to be periodic in nature. The merits of such a technique, as mentioned by the researchers, is that deep packet inspection is not necessary, as they focus on analyzing timing data from Network Flows, thus bypassing Law restrictions, concerns of privacy and the time-consuming process of inspecting all collected packets. Additionally, this method does not require knowledge of when a Network Flow occurred, but how long it lasted, rendering the process of trying to convert from a one-time system to another obsolete.

Bou-Harb et al. (2017) [114] focused on distributed malicious events that take place in vast areas of the internet, dubbing them campaigns, with their goal being the development of efficient techniques to analyze vast quantities of network traffic and produce network forensic evidence. However, this approach focused on the identification of probing botnets alone, which means that other botnets that do not employ such mechanics will not be detected. Additionally, as mentioned by the researchers, this approach suffers from poor scalability, that is, when the number of infected machines probing the system exceeded 1000, both false positive and false negative rates started to increase. A drawback of using network flow analysis, compared to other methods such as Deep Packet Inspection, is that it bypasses the payload of packets which means that certain information is ignored. Thus, it can't be used for malware and command extraction from traffic and some attacks that rely on the payload such as SQL injections can pass undetected.

Sivaprasad et al. (2018) [112] proposed a monitoring system which relied on machine learning techniques applied to network flow-summarized data. The researchers used a combination of data from the CTU-13 dataset and packets that were collected from a DDoS attack they performed. Their task was to create a user-friendly interface, where data was uploaded to the system, pre-processed with Weka and after the feature-extraction process, Naïve Bayes and SVM classifiers were built. On a similar note, Mathur et al. (2018) [117] proposed a model for botnet traffic discrimination. The researchers investigated the predictive capabilities of five different classifiers, *randomized filtered classifier, logistic regression, random committee, random subspace, multi class classifier*. After features were extracted from a combination of data from CTU-13, ISOT and live captures, the five classifiers were trained and compared. It was then shown that the logistic regression and multi-class classification algorithms had the best performance in both accuracy and false-positive rates. This work is a good indication of the relative performance of various classifiers when tasked with botnet identification, even though neural networks were not included.

Kozik et al. (2018) [118] used a distributed Apache spark environment to train an *Extreme Learning Machine (ELM)* classifier. ELMs differ from other neural networks, as they often include a single hidden layer, and have different activation functions. The key contribution of this work was that the distributed environment was incorporated in the training of the ELM classifier, by splitting calculations performed at the hidden layer into chunks of computation which could be performed by the distributed environment's elements. Results showed that the proposed method is promising.

Pektas et al. [119] proposed a deep learning system to process network flow patterns and identify botnets. In a botnet, communication between C&C and bots is frequent, as such their approach was to target these channels. Their choice of deep learning was justified, as their method relies on processing large quantities of data, in which deep learning thrives. During feature extraction, collected flows were turned into graphs, grouped by communication endpoints, which allowed them to produce new statistical features. A number of different configurations were tested, with varying numbers for layers and neurons. The researchers concluded that deep learning presents acceptable accuracy for botnet identification in flow data, with the added bonus that feature selection is not necessary, as deep networks identify the best features.

Amini et al. [35] proposed a combination of X-mean clustering and rule-based classification for the netflow-based recognition of centralized HTTP botnets. In the proposed system, routers collected netwflow data which was stored in a central database. Next hierarchical clustering on *protocol, source and destination IP* followed by x-means clustering produced highly similar groups of flows. Finally, rules-based classification based on entropy was performed to identify flows that occurred at regular intervals and were not produced by legitimate protocols. Results reported by the researchers indicate an accuracy of detection of over 95%.

Le et al. [120] investigate the applicability of Self Organizing Map (SOM) unsupervised learning, towards the identification of botnet activities in the wild. By using three publicly available datasets CTU13, ISOT and HTTP-CSIC the researchers employed SOM, an unsupervised neural network which uses competitive learning and results in dimensionality reduction. Three strategies were used: mix of normal and attack, only normal and only attack traffic, to evaluate SOM's performance. Classification in the first scenario is done through a majority vote of the nodes, while for scenarios 2 and 3 is outlier threshold-based. Results indicate high accuracy of distinction between botnet and normal traffic, over 99.5%.

- **Deep Packet Analysis (also known as Deep Packet Inspection or DPI)-** [45, 121, 122]- is a form of packet filtering, that relies on the inspection of both headers and the data segment of a packet, for the purpose of identifying malicious traffic based on known patterns. Although it raises privacy concerns, faces problems when trying to parse encrypted traffic and relies on a signature database to perform its identification, thus being unable to identify Zero-day malware, DPI is still used to this day. By scanning the content of packets, more information can be gathered, and the behaviour of the packet's origin can be better understood.

Chen et al. [121] proposed and implemented a cloud-based collaborative network security management system, which takes advantage of Cloud storage and processing, to perform offline forensic operations on captured raw network traffic with emphasis given on SPAM incidents. The research team utilized the Collaborative Network Security Management System (CNSMS), which managed the security of four different sites (networks), by making use of NetSecu nodes and Probers to gather, monitor and manage network traffic. The NetSecu nodes have the advantage of interacting with locally deployed security mechanisms (such as firewalls and IDS), dynamically responding to security incidents threatening the network. Additionally, they can be integrated with self-protection solutions, in the event they become the target of an attack. Furthermore, NetSecu nodes communicate with similar devices deployed in other networks, thus creating an overlay network. It was observed that this cloud-based scheme could be applied to the investigation of other network-related security incidents.

Cheng et al. [45] developed D^2PI, a system that identified malware in network traffic, by using a Deep Neural Network. The proposed system was a Convolutional Neural Network (CNN) which classified collected traffic, into either "malicious" or "benign", based solely on the payload. After extracting the payload, their length was regulated to a predefined length and incorporated in a matrix, in order to be processed by the CNN. Results indicated that, although more work is needed to improve this method, it is a promising first step towards incorporating CNNs in DPI systems.

Another mechanism that has been leveraged in DPI systems, is *finite state automata*. Finding ways to improve these mechanisms, in order to be able to handle the ever-increasing variety and volume of traffic was the focus of Yin et al. [122]. As regular expressions, which can be used to identify complex patterns in packet bodies, are often implemented in finite state machines, improving their efficiency is of vital importance. Initially, the researchers discussed deterministic and non-deterministic finite-state automata, comparing the memory requirements for each category. They concluded that a non-deterministic approach is needed for a DPI implementation, and provided improvements that help reduce processing time and memory consumption. Experiments s that were made, between two automata, based on regular expressions from Snort, showed that a non-deterministic finite-state automaton machine with the proposed improvements used less memory, as the number of conversion edges was reduced.

Holkovič et al. [123] developed a rule-based method for automatically detecting incident-patterns in network traffic. The system takes as input event descriptions, written by a network administrator in a human-readable language and pcap files that have been converted to an easier-to-parse format. It then loads the event descriptions and scans the converted pcap files to identify events which are displayed in a final report. This tool is designed to enable network administrators to investigate network incidents, without requiring the experience or the time of a manual investigation.

- **Attack recognition**- [124, 125, 126, 127, 128]- is a collection of machine learning techniques applied to any number of sources (e.g., network traffic and logs). It is utilized in investigating the identification of known patterns exhibited by malicious software. Although pattern identification is generally used in some form or another in different techniques, in this section, such patterns are identified not only in packets, but from other sources as well, like network logs, and are used to better understand the sequence of events which lead to an attack. This forensics method, in some situations, requires access to the physical device, from which logs and files are extracted and then examined offsite.

  In their approach, Zhu et al. [127] developed an algorithm, that identified a sequence of attack events, by scanning network logs. Their algorithm identified what the research team dubbed "attack bubbles", with high suspicion values. An "attack bubble" was defined as a tuple containing: a collection of network events found in the logs, a suspected type of attack which these events might constitute, a probability that the identified events constitute the suspected attack, and an identification of the source of these events (IP address). Han et al. [125] proposed a technique that examined the communication behaviour of network nodes. In their process, they focused their efforts on identifying Command and Control communication which exhibit a pattern of synchronicity, for instance, the C&C server sends instructions to all botnets simultaneously, and all bot respond at the same time.

  Karthika et al. [126] proposed a highly scalable system for detecting stealthy peer to peer botnets, akin to an Intrusion Detection System. Their system identified all P2P (Peer-to-Peer) traffic in a network, relying on DNS lookups which the system collected, and reasoned that most P2P applications do not rely on DNS to establish the destination IP address in legitimate P2P communication. A system for investigating the existence of botnets, composed of several modules was proposed by Bansal et al. [124]. This system gathered all network data that are sent or received by the internal network, creating a repository of stored packets, which were later filtered, flagged as legitimate or illegitimate traffic and then used to identify the presence of any unknown botnets. The process would then scan any involved systems, to gather traces that were not identified by previous steps. In the final step, a report was generated, with all identified evidence being visualized.

Kim et al. [128] proposed a system that detects hacking attacks by constructing a tree structure and can be utilized in real-time. The system is separated into a *pre-phase* and a *post-phase*. In pre-phase, data is collected, normalized and an attack tree is constructed. In post-phase, collected logs are analysed, and either the user is notified or the system is shut down. Experiments indicated quick real-time response to attacks, detecting a backdoor attack in 687 ms.

- **Visualization of Network traffic**- [129, 130, 131]- a number of diverse visualization techniques have been employed, to improve Botnet investigations. One use of such techniques is as a support tool for investigators, which can assist security experts to track the route taken by a malware infection carried out at large workstation areas.

  Such a visualization method was developed by Joslin et al. [130], who studied the representation of Network Flows (or IPFlows) as a directed graph, combined with relational information, making the argument that the proposed combination motivates the creation of new hybrid graph-relational systems. Concentrating more on the visual representation of network traffic and security incidents, Gugelmann et al. [129] produced Hviz, a traffic visualization program that processes HTTP/S traffic in order to reduce the number of events that an investigator would have to work on. They employed various mining techniques (FIM) to aggregate the sites visited by workstations during an investigation. Also, by comparing traffic between workstations, they attempted to figure out if the traffic in question is malicious or not. On the other hand, an investigation of UDP flooding attacks, by using the UDP flow graph was performed by Anchit et al. [131]. The researchers made use of a testing environment, set up in their lab, choosing Wireshark for the collection of network traffic. As stated by the researchers, this approach could assist investigators in identifying network attack incidents, as patterns of network communication become easier to spot with the naked eye.

- **Intrusion Detection Systems-** [99, 100, 101, 102, 132]**-** are generally a defensive mechanism deployed either in the network (Network IDS) or in a device (Host IDS), which can either employ pre-made signatures (signature-based IDS) or machine learning (anomaly-based IDS). In the context of Network Forensics, IDS systems can function as alarm triggering mechanisms which, after identifying malicious activities (for example Botnet traffic), can raise further forensic mechanisms, allowing for an automated solution.

  AlRoum et al. (2017) [132] developed a Botnet Detection System that focused on DNS records, as some botnets harness DNS communication in order to make their Command and Control infrastructure more resilient and avoid detection. Their solution relied on seven factors, *domain reputation, geo location, destination port, known C&C, domain owner, frequent DNS changes and behavior.*Weights were assigned to each factor, the sum of which would produce a DNS flag, based on which an alarm would be raised. Furthermore, the seven factors were further partitioned into two groups, the must-stop factors and the partial-stop factors. If one must-stop factor or three partial-stop factors were detected, then a flag would be raised, indicating a suspicious domain record. The research team reported an accuracy of close to 63% when they tested their solution against similar results derived from the cybersecurity company FireEye.

  In the field of anomaly network IDS, Aldwairi et al. [133] investigated the applicability of Restricted Boltzman Machines (RBM), in order to distinguish between normal and abnormal flow traffic. An RBM is a special kind of neural network, where layers are either visible or hidden and two layers of the same type can not be connected. In their experiments, a balanced subset of the ISCX dataset was used to train the model. The algorithms that were used to train the RBM, were constructive and persistent constructive divergence. Results showed that RBMs are a valid choice for an anomalous network IDS with the capability to identify novel abnormal traffic. On the other hand, the performance of several supervised classification models for network IDS was investigated by Ugochukwu et al. [134]. Four classifiers were tested, Naive Bayes, C4.5, Random Forest and Random Tree. Out of the four classifiers, Random Forest and Random Tree outperformed the rest. In a different study that focused on clustering algorithms, Qi et al. [135] investigated the utilization of an improved K-means for real-time intrusion detection systems. The researchers enhanced the algorithm by adding new steps that were used during the process of determining the centres of a cluster, like cross-entropy between data points. The improved K-means was then tested on the KDD99 dataset, with results indicating that it outperformed the traditional K-means implementation, with an accuracy of 97%.

B. Network Forensic methods for investigating Botnets in the IoT

In this section, we focus on network forensics methods, that were designed to be applied in an IoT environment. One might consider, that pre-existing network forensics mechanisms could be employed in the IoT, with the same accuracy and efficiency as when applied to conventional computing systems. The fact is that the quantity and speed with which data is produced in the IoT, as-well-as its diversity, require the development of new methods which take under account these characteristics of the IoT. Popular methods and recent studies are explained as follows.

- **Honeypots**- [136, 137, 138, 139, 140, 141]- which would be an ideal decoy for malware that target IoT devices was developed by Yin Minn Pa Pa et al. [136], named IoTPOT. Their proposed system was a combination of a low-interaction front-end interaction program, and a high-interaction back end system, named IoTBOX which is a collection of virtual IoT machines (Linux OS), that help make the Honeypot appear as a legitimate, more dynamic device. In this design, they also incorporated a Profiler, which stored incoming "malicious" commands and the responses produced by IoTBOX. This would allow the system to produce the appropriate response in future interactions without invoking IoTBOX. Another module that was used was a Downloader, which managed all downloads prompted by the attackers and a Manager which handled the configuration of the system. It is mentioned that the virtual environment required manual OS image resets from time to time, a process which could possibly be automated. It should be mentioned that some malware include anti-forensics capabilities which can thwart attempts to scan them in a virtual environment, which was not discussed in this work.

  With the intent to produce an initial framework for a high-interaction, seemingly geographically-distributed and vendor/type-of-device diverse Honeypot, Guarnizo et al. [137] proposed SIPHON. Their implementation allowed for the deployment of more than 80 high-interaction virtual IoT devices, with their IP addresses being distributed around the globe, and having only

7 physical devices exposed. The projected scalability of this system was reported to be i*w, with 'i' being the number of physical IoT devices and 'w' being the wormholes in use. Collected data from a two-month period, showed a significant amount of incoming attack traffic which targeted SSH services that were exposed by the experimental system. Additionally, the team noted that the proposed honeypot was not identified as such by Shodan, making this framework a viable solution.

An IoT-based honeypot, which focused on the emulation of an entire IoT platform was developed by Wang et al. [138], named ThingPot. Their design was an open-source project, that could be characterized as a "Middle Interaction Honeypot" which made use of both high and low interaction modules. The proposed framework of ThingPot included three groups of entities, Extensible Messaging and Presence Protocol nodes (client, server) used for communication between "user" and Controller, REST API which represented the IoT device that ThingPot was mimicking, Controller which was represented as a PC that gathered log files from the other nodes in the setup. To test the applicability of ThingPot, an arrangement of devices which ran on Raspberry Pi, connected to a PC, and simulated Philips Hue lightbulbs was setup. This implementation was, as reported by the research team, a proof of concept that focused solely on the Philips Hue lightbulb IoT platform, with support for other such devices pending. Also, the research team's experiments ran for 1.5 months, a possible extension of this time could have yielded different results.

In the field of military network security and operations, Hanson et al. [139] introduced the concept of "Honeyman", an IoT honeynet architecture that, instead of functioning as a mitigation and attack analysis platform, its primary function would be to provide indication and warning as-well-as distributed deception capabilities. The proposed system's function would primarily be to deceive attackers about the location and status of military IoT devices, whose goal was to corrupt their intelligence about either geospatial or system data. A multi-tier architecture was proposed, where a combination of light-weight devices would be used in tandem with virtualized machines and a software-defined network, gathering data from the attacks. All collected information would then be forwarded to an RNN-based analysis module, where inferences could be made about the motivation of an adversary. Several difficulties hinder the development of such a system, with two being discussed in this research being: *securely emulate embedded os communications*, *avoiding detection of emulated environments.*

A server-based IoT honeypot system was proposed by Gandhi et al. [140] and named *HIoTPOT*. The proposed system relied on a Raspberry Pi to act as the "middle-man", and divert users with unknown credentials, with such attempts being recorded in a database, to a virtualized image of the real IoT devices. From there, alerts would be sent to legitimate users and logs would be created, that record the attacker's interaction with the environment. The proposed system was shown to provide more interactions as-well-as new mechanisms than an existing one, further comparing their performance and showing that HIoTPOT outperformed the existing solution in packet loss, consumption of bandwidth and added delays.

The problem of developing realistic IoT honeypots was addressed by Luo et al. [141]. The researchers proposed an automated method for crafting low but intelligent-interaction honeypots. To build such a honeypot, attack requests were collected by an initial honeypot, which were then forwarded to a specialized module that probed live IoT devices to get legitimate responses. In this way, the honeypot would be able to gather relevant responses from real devices. Machine learning would then be applied to the collected responses, in order to craft a "profile" that best represents the IoT device that the honeypot would mimic. The evaluation of the proposed method indicated an improvement in the functionality of the honeypot, extending its interaction time with attackers. Shrivastava et al. [142] worked on capturing attacks targeting IoT devices by using Cowrie, a medium interaction honeypot. The Cowrie honeypot monitored the attacker's behaviour, storing their actions into logs, from where the attacks were separated. Feature selection is carried out through wrapper methods in Weka. The problem of distinguishing between the attack types was formulated as a multi-class classification problem and an SVM classifier displayed the highest accuracy at 97.4%.

- **Network Flow Analysis**- [33, 143, 144, 145]- the research team of Galluscio et al. [145], having the intent to clarify the severity and magnitude of IoT infected devices worldwide, worked from an empirical point of view. They utilized unsolicited darknet-generated data, which by definition, imply a potential malicious scan. As such, and to be able to identify scanning (otherwise known as probing) activities, they developed an algorithm which utilized network flow features. The proposed algorithm compared the observed packet count and rate of a flow within a set time window, to pre-determined packet count and rate values. If observed values exceeded the pre-determined thresholds, then the flow was flagged as a malicious scan. As the research team was interested in IoT infected devices, they then made use Shodan, to infer whether the identified malicious probe originated from an IoT device, which would imply that that IoT device was compromised. Their findings showed that IP cameras and routers were the top two most heavily exploited household IoT devices, while sectors like Manufacturing and Building automation had the largest portions of exploited devices. The proposed algorithm, although fast is simplistic in its nature, with the assumption behind it being that benign Darknet IP devices don't perform Internet-wide scans.

Being able to identify consumer IoT devices that are part of network attacks is an important task. Towards that goal, Doshi et al. [33] worked on the identification of IoT devices which took part in launching DDoS attacks. Their approach was to utilize several characteristics of IoT-generated traffic that distinguishes it from other non-IoT traffic, such as the frequency of communication. As such, through an initial feature engineering process, the researchers trained five machine learning models, including a neural network, and concluded that real-time detection of DDoS originating from IoT devices is possible.

Akin to the previous study, Meidan et al. [144] developed a method that identified botnet membership in commercial IoT devices. Their proposed method was based on deep autoencoders, one for each of the nine IoT devices used in their experiments, with the life-cycle of their method being:

data collection, feature extraction, training of autoencoder model, continuous monitoring. The principle behind their work was that IoT devices have a finite set of states, and as such, their autoencoder was trained to model normal traffic, flagging its errors as abnormal/bot activities. By testing their approach against real IoT botnets Mirai and Bashlite, they demonstrated an FPR of close to 0 for the majority of IoT devices under scrutiny. On a similar note, Nguyen et al. [143] developed DIoT. In this system, data was first gathered from on-line IoT devices, and fingerprints of device communication were derived from them. This system, utilized an unsupervised technique to create clusters of fingerprints of IoT devices, and distinguish between different device types and models. An anomaly detection module, that implemented a k-Nearest Neighbors classifier, identified abnormal traffic, which worked as an indication of a compromised IoT device.

Monge et al. [146] presented FlowSentinel, an approach for DDoS attack membership identification in IoT networks. Originally designed for android environments, it was adjusted to handle heterogeneous data so that it could be applied to IoT networks. Its functionality is based on Self Organizing Networks (SON). The focus was on detecting flooding DDoS attacks from the source-side. The FlowSentinel process involves the use of a Random Forest Classifier for the identification of the best method to detect the attacks. Results indicate that FlowSentinel is a promising solution to the aforementioned problem

- **Intrusion Detection Systems**- [147, 148, 149, 150, 151]- Roux et al. [148],
  - created an intrusion detection system for IoT, which focused on identifying potential attacks, based on their relative position in the environment which was monitored by this IDS. Their design made use of wireless sensors, strategically positioned around the premises (house), which were tasked with gathering information regarding signal strength and direction. The gathered information would then be forwarded to a central device, where it would be processed by a neural network that has been trained to identify legitimate transmissions from legitimate positions inside the network. Any transmission that originated from outside the network, would then be flagged as illegitimate, causing alarms to be triggered. Such a technique can be used to counteract war-driving and war flying, which can be employed to infect IoT devices with bot malware. A possible extension of its functionality would be to use the flagging mechanism to automatically trigger forensic solutions for the IoT when such illegitimate transmissions are identified.

  Although not explicitly stated, Al-Dabbagh et al. [147] proposed a framework for designing distributed IDSs of an IoT-like wireless control network. The proposed distributed IDS consisted of individual IDSs in each node/actuator in the control network, with the network itself modelled as a linear time-invariant system. This allowed for the identification of cyber attacks in a neighbouring group of nodes of the network.

  Abhishek et al. [149] introduced a centralized IDS for IoT clusters. The researchers identified the gateway of an IoT cluster as a weak point, and thus they focused their efforts on monitoring the gateway. The novelty of the proposed IDS is that focus was placed on the downlink channel of the gateway. The proposed IDS identified malicious gateway attacks that sought to corrupt packet integrity, thus forcing retransmissions and taxing battery life. These attacks were identified by investigating the packet drop probability of the downlink between each IoT device and its gateway. In the future, the researchers intend to extend their work by studying the uplink packets, as-well-as different types of attacks.

The process of active learning for the development of IDS in wireless IoT networks was investigated by Yang et al. [150]. The concept of active learning relies on training a model on a small group of unlabeled data and periodical re-training, by requesting the missing labels of a record from a human operator. The proposed method relied on an initial detection of outliers by using an unsupervised outlier detector followed by the application of the active-learning-based scheme. During the active-learning-base learning process, first supervised learning was employed, followed by label selection and finally labelling by the expert operator, with the process being repeated until precision and recall reach appropriate values. Still, challenges exist in this field which affect active-learning, such as the constrained power of such devices.

An ensemble NIDS was proposed by Moustafa et al. [151]. The researchers focused on identifying botnet attacks targeting the DNS, HTTP and MQTT protocols. Statistical methods were used on data, to produce additional features that improved the classification process. Feature selection was performed through the calculation of the Correlation Coefficient value between features. The AdaBoost ensemble method was implemented, using Decision Tree, Naive Bayes and Artificial Neural Network classifiers. Results showed that the proposed method outperformed equivalent existing ones in processing DNS and HTTP flows.

2.5. Deep Learning and its role in Network Forensics

Artificial Neural Networks (ANNs) which were inspired by the inner mechanics of the human brain, specifically the underline interconnected networks of neurons, are a type of machine learning technique which convert input data into output by employing non-linear transformations. ANNs can be roughly grouped by the number of layers that make up their architecture (excluding input layer), into *shallow* and *deep* [152]. Although there exist no strict definitions for them, a shallow ANN typically has one to two layers, while deep ANNs can have hundreds [52]. With the wide adoption of deep ANN architectures in various fields (e.g. computer vision, pattern recognition, . . .), new specialized architectures have emerged.

Deep NNs can be further classified based on the way that they view the data and the classification problem, as given by Hodo et al. [153]. These two groups are *discriminative* and *generative* models. Discriminative models are supervised methods tasked with separating the data into classes by focusing on the decision boundary of the classes and calculating the conditional probability of the class feature, with respect to the data features (P(Y/X)). Prominent examples include:

- **Recurrent Neural Network (RNN)-** as a discriminative model, an RNN can be useful when the information maintains some temporal relation to its previous states.
- **Convolutional Neural Networks (CNN)-** is a type of space-invariant Multilayer Perceptron, inspired by the interconnections present in the visual cortex of the brain. It is comprised of multiple hidden layers such as convolutional layers, pooling layers, fully connected layers and normalization layers.

On the other hand, generative models are considered to be unsupervised, as they do not require labelled data, and instead calculate the joint probability of data and class features (P(X,Y)) and build models that best describe each class separately. Some prominent examples of such models are described below.

- **Deep Auto Encoder (DAE)-** is a type of NN that is used to learn efficient data coding in an unsupervised manner. Typically, it includes an input layer, multiple hidden layers and an output layer of the same size as the input layer, where the input data is reconstructed.
- **Deep Boltzmann Machine (DBM)-** produces binary results by relying on stochastic units and energy states. A restricted Boltzmann machine (RBM) is comprised of a visible input layer and a single hidden layer. By stacking multiple RBMs, so that the hidden layer of one produces the input for the next, one can build a DBM.
- **Deep Belief Network (DBN)-** are networks of interconnected layers comprised of multiple stacked RBMs. Again, connections between nodes of the same layer are not allowed, similar to DBMs. Training a DBN in an unsupervised manner requires for each layer to be greedily trained.
- **Recurrent Neural Network (RNN)-** is a type of deep NN that can be trained either as a supervised or unsupervised model. The main difference

between RNN and a deep Multi-Layer Perceptron would be that the RNN maintains an internal memory of previous calculations performed inside the network. Hidden layers of RNN “feed” information that is used for the next iteration of the algorithm.

Multiple deep learning solutions have been proposed for application in the field of Network Forensics in recent years [34, 154, 155, 156, 157, 158, 159]. Yin et al. [157], proposed a Recurrent Neural Network-based IDS which outperformed other classifiers used for the same purpose. Similarly, Kang et al. [154], proposed an IDS for a vehicle network capable of performing in real-time, with an average accuracy of 98%. In work by Zhao et al. [158], a Deep Belief Network was first applied, to reduce data dimensionality, followed by the training of a probabilistic neural network. Shone et al. [159] combined non-symmetrical auto-encoders with a random forest classifier to classify network traffic from the KDD99 and NSL-KDD dataset, with results indicating an increase in accuracy when compared to DBNs.

Niyaz et al. [155] used stacked auto-encoders in their implementation of a DDoS detection system for software-defined networks. The multiple auto-encoders were greedily trained layer-by-layer, with the output of one layer being the input of the next. Then the entire network was fine-tuned as a classifier. Reported accuracy for distinguishing between normal and attack traffic was 99.82%, outperforming other classification methods such as shallow NN, while individual types of DDoS attacks were identified with an accuracy of 95.65%. Lotfollahi et al. [156] used a combination of a one-dimensional CNN and stacked auto-encoders for automatic feature extraction and classification of network traffic, achieving both application identification and traffic characterization in either encrypted or unencrypted traffic.

2.6. Inherent challenges in Network Forensic Investigations of IoT Botnets

The process of designing IoT protocols and sensors and the lack of standards are the main reasons why the IoT is an easy target for botnets. This gives rise to many challenges for experts who intend to investigate such security incidents[160, 161,

Table 2.3: Most vulnerable communication Layers in IoT systems

Communication layers	Description
Link layer.	It is for device discovery and local communications, at the link layer, the most widely used protocols include RFID, Wi-Fi, Bluetooth, ZigBee and Z-Wave [3, 4]. However, having in mind the global and, sometimes distributed nature of the IoT, the most significant protocols to mention, from the scope of a Network forensics investigator, would be the ones located at the network layer.
Network/Transport layer.	The most commonly used protocols are IP (v4, v6) [3, 4], which are widely used by most devices in the Internet, UDP, a Transport Layer protocol ideal for real-time communications, with light-weight alternatives also in use, such as NanoIP [4], a TCP/IP -like stack of protocols, appropriate for embedded devices.
Application layer.	It is one of the most widely used protocols, include HTTPv2, REST and SOAP, which handle communications between applications (Client-Server systems).

162, 163, 164]. We discuss the main challenges that could inhibit network forensic investigations of botnets in IoT systems, as explained in the following section.

- **Interoperability-** there are constraints in the interoperability of the IoT. As no single set of standards and specifications have been widely accepted [6, 163], every vendor implements their IoT products differently, choosing technologies, operating systems and protocols to serve the needs of their products, which often require a Sensor Bridge to co-exist, as shown in Figure 2.4. Moreover, we describe in Table 2.3 the most vulnerable communication layers and protocols in this architecture. It is obvious that the lack of specifications causes problems to the development of a single forensics solution that is capable of handling a family of IoT systems and devices.

- **Forensic soundness-** with the IoT designed to work in an autonomous and ubiquitous form, following a forensically sound process becomes a challenge [164]. Preserving the scene of a crime where IoT devices are involved is challenging, as data is in constant motion and the scope of the investigation

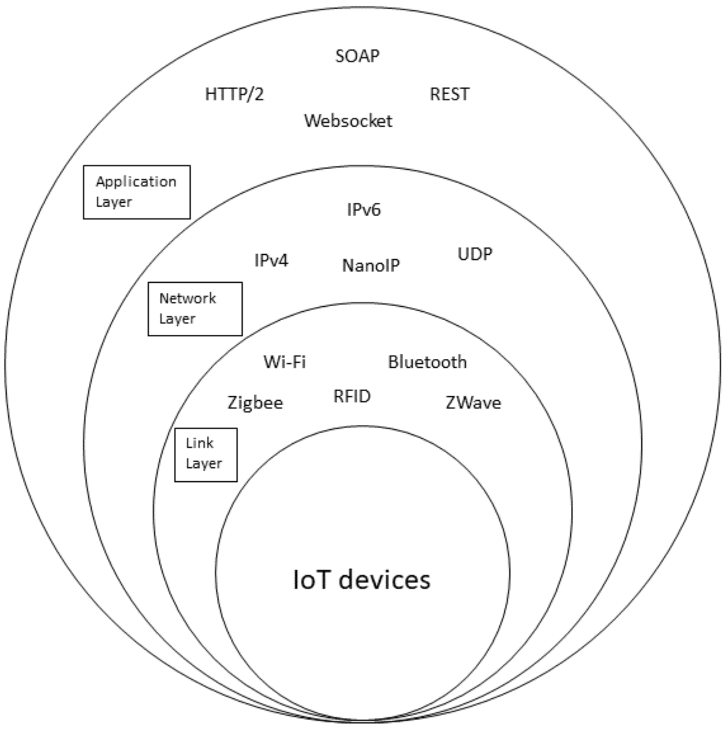


Fig. 2.4: Architecture of IoT [3, 4]

is not clear. There is a lack of documented methods and reliable tools for collecting evidence in a forensically sound manner. As most IoT devices don't retain metadata that can indicate alterations or manipulations of files, and the time when these changes occurred, correlation of evidence between IoT devices is challenging. Finally, without a forensically sound monitoring system, attribution becomes difficult.

- **Big Data characteristics generated from IoT systems-** some of the challenges present in investigating the IoT, coincide with the main characteristics of Big Data, indicating that the latter technology could be a possible approach to handling these challenges [163]. Two of these Big Data characteristics are Velocity, Volume as discussed below.

  - **Velocity-** the increases in Internet speed over the years, ensures that communications are nigh-instantaneous, which functions as one of the enabling factors of the IoT. This translates to a large number of constantly functioning, small sensors and actuators ("smart things") sending collected data and feedback, at high speeds, to the service providers (usually located in the Cloud), which in the context of IoT botnets means that such botnets will possess an army of high-speed and always available bots. With data and evidence produced fast, and having a relatively short life in the network, a need to analyze data in real-time (or as near as possible) is essential, if the results produced by investigations are to be of any real value.

– **Volume-** with more than 8 billion IoT devices deployed in 2017 and projections for the near future rising even higher, it is evident that data produced by the IoT will also skyrocket. As such, having many small embedded devices in constant use produces huge quantities of data that, in the context of forensics (regardless of the type of investigation), will inhibit the effectiveness of investigations, burying useful traces and evidence under a sea of noise (in the form of collected data). On top of that, an increasing number of deployed devices equates to an enhancement of numbers for potential hijacked Bots, allowing adversaries to take advantage of the sheer volume of data that the IoT can produce and thus launch massive and reliable Cyber-attacks (example Mirai [68]).

In the work presented in this thesis, we provide solutions to the aforementioned challenges. The Particle Deep Framework (PDF) presented in Chapter 5 addresses the interoperability challenge, as it utilizes the TCP/IP protocol suite to scan traffic and most IoT solutions transmit their collected data or receive and the forensic soundness challenge through the use of cryptographic hashes and a fingerprinting process. The Big Data challenges of velocity and volume are represented through the creation of the Bot-IoT dataset in Chapter 3 and addressed by the PDF in Chapter 5 through the use of deep learning, a machine learning method specifically designed to work well with large quantities of data and in short periods of time.

2.7. Conclusion

In this chapter, we explored the effects that the expanding IoT domain has had in Network Forensic Investigations of IoT botnets. We initially provided background for the Internet of Things, botnets and Digital Forensics, as a foundation. We give a new definition for the IoT, which places the interconnection of "Things" and their service-like functionality at the forefront. We argue that Deep Learning is a viable solution to handling the types of data produced in the IoT, and thus discuss its applicability in Network Forensics. Furthermore, we provided a taxonomy of Network Forensic mechanisms which could be applied to botnets in both non-IoT and IoT environments, including their strengths and weaknesses. The Network Forensic mechanisms that were discussed, were Honeypots, Network Flow Analysis, Deep Packet Analysis, Attack Recognition, Visualization of Network Traffic and Intrusion Detection Systems. Several challenges were presented and their solutions discussed and further expanded upon in the rest of this thesis. In Chapter 3 we present the Bot-IoT dataset, its creation process and feature generation. The machine learning analysis of the Bot-IoT dataset is provided in Chapter 4 and in Chapter 5 we introduce the novel network forensic framework called Particle Deep Framework. Finally, the concluding remarks are given in Chapter 6.

Chapter 3

Development of the Bot-IoT dataset and its statistical analysis

3.1. Overview

The rapid development of the Internet and the emergence of the Internet of Things (IoT) have attracted the interest of cyber attackers for exploitation through various complex hacking techniques such as Botnets. This has been compounded by the lack of standardization in IoT systems, as well as in the cheap, lightweight and low-powered devices that comprise many of these systems as described in Chapter 2. One way that IoT networks have been exploited for criminal purposes is in the propagation of Botnet malware, which has been shown to launch DDoS of up to 1.1 TBps [12, 165].[1]

With new and unique threats capable of compromising IoT networks, and as existing techniques are inadequate to address them, it is important to develop advanced forensics methods for identifying and investigating adversarial behaviours. Network Forensic techniques are widely used for analyzing network traffic and to identify infected devices taking part in major cyber-attacks [166]. Additionally, due to the number and nature of IoT devices, the workload of processing collected data would be an ideal application of Big Data analytics [167]. Big Data analytics is a collection of sophisticated analytical processes, which were designed to handle three main problems, variety, velocity and volume [168]. Since IoT networks generate enormous volumes of data, it is imperative to employ analytical techniques capable of handling them, in a near-real-time fashion. As such, the term *forensic analytics* is defined to demonstrate the role of forensics techniques and big data analytics.

Forensic analytics demand big data sources for validating their credibility in IoT networks. In order to develop forensics and intrusion detection solutions that

[1]The work presented in this chapter has been published in:
Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, Benjamin Turnbull, Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset, Future Generation Computer Systems, Volume 100, 2019, Pages 779-796, ISSN 0167-739X, https://doi.org/10.1016/j.future.2019.05.041.

identify and investigate cyber-attacks, the development of a realistic dataset is still an important topic of research [31]. Over the years, a number of datasets were developed, each of them having its own advantages and disadvantages. Existing datasets, although applicable in some scenarios, introduce various challenges, for example, a lack of reliably labelled data, poor attack diversity such as botnet scenarios, redundancy of traffic, and missing ground truth [57, 58, 59, 60]. However, a realistic Botnet traffic dataset in IoT networks has not been effectively designed. The new Bot-IoT dataset addresses the above challenges, by having a realistic testbed with multiple tools being used to carry out several botnet scenarios, and by organizing packet capture files in directories, based on attack types.

The main contributions of this chapter are as follows:

- We design a new realistic Bot-IoT dataset and give a detailed description of designing the testbed configuration and simulated IoT sensors.
- We then statically analyze the proposed features of the dataset using Correlation Coefficient and Joint Entropy techniques.
- We also evaluate the performance of network forensic methods, based on machine and deep learning algorithms using the Bot-IoT dataset compared with popular datasets.

Chapter 3 is structured as follows. The literature review of the study is discussed in Section 2. In section 3, the testbed that was used to develop the Bot-IoT dataset is presented. Sections 4, 5 and 6 discuss the feature extraction process, the benign and malicious scenarios and the statistical and Machine Learning methods used to analyze the dataset respectively. Finally, in section 7, experimental results are presented and discussed, followed by the conclusion of this chapter.

3.2. Justification for the need of the new Bot-IoT Dataset

This section explains the IoT and forensic techniques-based machine learning that are used in this work. Additionally, information is given about popular network datasets and their limitations for designing forensic techniques in the IoT.

3.2.1. IoT and forensic analytics-based machine learning

IoT networks consist of both typical network elements (including workstation, laptops and routers) and IoT devices. Machine-to-machine technologies used in conjunction with cloud paradigms, provide the resulting IoT services to users and organizations, in geographically distributed locations [74]. Popular applications of IoT systems, such as smart cities, smart healthcare and industrial IoT, are complex because they have multiple sensors that need significant processing power and computation to allow individuals to control the IoT elements in large-scale networks [74].

The vulnerabilities of IoT networks significantly increases with complex cyber-attacks, such as Botnets. Botnets are synchronized networks of compromised machines, called bots [82, 92]. In a Botnet, the attacker, called *botmaster*, maintains a bi-directional channel of communication towards the compromised network, through a Command and Control (C&C) infrastructure. By using the C&C a botmaster issues commands to the rest of the botnet and is capable of receiving feedback regarding attacks in progress or even stolen data [82, 92]. A botnet can launch a number of activities, such as Distributed Denial of Service attacks (DDoS), Keylogging, Phishing, Spamming, Click fraud, Identity theft and even the proliferation of other Bot malware [22]. The compromised bots are under the complete control of the botmaster.

Digital Forensics is defined as “the identification, preservation, collection, examination, analysis and presentation of digital evidence” [105, 169]. During this process, forensic methods are applied to data (or in this case, big data), in order to draw conclusions about the occurrence of a crime, including who did it, their motives, what methods they used and what, if any, information or machines they affected. Since the forensic process is linked with analyzing big data to design effective techniques in law enforcement, such techniques can be utilized using Machine Learning (ML) algorithms for designing reliable models that can efficiently determine cybercrime. Therefore, we entitle forensic techniques that use Machine Learning and big data as “forensic analytics”. Forensic analytics could be used in the examination phase, where the Forensic analyst seeks to identify patterns that would answer the aforementioned questions, related to the occurrence of a crime.

Network Forensics analytics is tasked with processing network data, such as logs, e-mails and network packets, the latter usually stored in the form of packet capture files. In literature, the common cyber applications of network forensics

-based ML are Intrusion detection, Honeypots, network flow analysis, deep packet inspection, and email authorship, as discussed in the following points:

Intrusion Detection Systems (IDS) - [170, 171, 172, 173] are classified into network and host intrusion detection systems. Network IDS (NIDS) are placed in a strategic point of the network, usually, a node connecting the internal network to the Internet and are tasked with scanning inbound and outbound traffic for known patterns of attacks. As known malicious patterns are identified, notifications are set off and subsequent forensic mechanisms can be deployed, thus lowering response time and potentially increasing the accuracy of investigation. Such IDS can incorporate ML, in their anomaly detection process [171], with classification, clustering and other methods used to detect unusual non-normal traffic. Examples of such systems have already been proposed [174].

Honeypots- are entities (software and/or hardware) that are designed to appear as an appealing, exploitable target to an attacker [174, 175]. Honeypots have several uses, to identify attackers that are able to bypass other control mechanisms, to mitigate ongoing attacks and distract attackers from real systems and collect attack binaries [175]. They are classified in three main categories, (i) Low Interaction Honeypots, where software simulates responses to certain stimuli, (ii) High Interaction Honeypots, where real Operating Systems are used as the honeypot, and (iii) Hybrid Honeypots a combination of (i) and (ii). In the context of honeypots, Machine Learning techniques are applied either to identify network traffic patterns that can be used to identify similar attacks in the future (by recording pcaps, collecting logs,...), or to analyse Malware samples [175] obtained during an infection.

Network Flow Analysis and Deep Packet Inspection - can be categorized into two broad groups: (i) Deep Packet Inspection and (ii) Network Flow Analysis [33, 166, 176, 177]. Both categories use captured traffic to draw conclusions regarding cyber-attacks. Their main difference is that (i) investigates the content of each packet, which yields more complete results but adds a considerable overhead whereas (ii) aggregates packets into flows, which are groups of packets that share *Source IP, Source Port, Destination IP, Destination Port, Protocol.* In both categories, Clustering, Classification and Association Rule Mining techniques have been applied in the literature.

Email Authorship- is an example of ML application in the context of Cyber Forensics, not directly related to Network Forensics. In such cases, Support Vector

Machines have been used to identify the author of an e-mail based on the text itself [178].

3.2.2. Comparison of proposed testbed with others

In this sub-section, we discuss the comparison between known testbeds found in literature, to the one we used to produce IoT-Bot dataset. A testbed is as an environment designed for the generation of network traffic [57]. A number of datasets have been introduced in the literature, to assist researchers in simulating Botnet activities and generating attack traffic datasets [32, 33, 57, 59, 179, 180, 181], as compared in Table 3.1.

Table 3.1: Comparison of testbeds, where (T=true, F=false),Types of traffic: 'A' for Attacks and 'CC' for Command and Control, and Large Botnet ('T' is greater than 10)

Testbeds	Virtual	Physical	Type of traffic	Attack variety	Large Botnet	Normal traffic	IoT traffic
Alomari et al. [179]	T	F	A	F	T	F	F
Carl et al. [32]	F	T	CC	F	T	F	F
Bhatia et al. [180]	F	T	A	F	T	T	F
Behal et al. [181]	T	T	A	F	T	T	F
Sharafaldin et al. [59]	F	T	A	T	F	T	F
Moustafa et al. [57]	T	F	A	T	F	T	F
Doshi et al. [33]	F	T	A	F	F	T	T
Proposed Testbed	T	F	A	T	F	T	T

To explain the benchmark datasets, various testbeds were used. To start with, similar to our approach, Alomari et al. [179] relied on a high-tier server, outfitted with Virtual Machines to implement their HTTP DDoS Botnet traffic testbed. Although their testbed employed a larger quantity of Bots (40) than ours, they limited their generated Botnet activities to only HTTP DDoS attacks, whereas we did not (description of activities in Section 5). The stated goal of this network testbed and the associated dataset was to produce traces of malicious traffic. Whilst this is important, the lack of full network packet capture makes it impossible to provide verification of any outcomes. Network traces also limits the data that can be extracted and processed.

Choosing to focus on the C2 (Command and Control) activities of a Botnet, Carl et al. [32], implemented their own IRC-centric testbed, in order to produce an ML approach to effectively identify malicious IRC traffic. In their work, they made use of a re-implementation of a real-world Botnet named “Kaiten”, which used IRC traffic as its C&C communication medium, and launched UDP DDoS from 13 “zombies” targeting a victim host. Contrary to their approach, in our testbed, we mixed normal and attack traffic by activating Ostinato [62] (for normal traffic) and the attack software at the same time, whereas their testbed was not tasked with generating normal traffic, with it being collected from a public network (later anonymised) and Botnet traffic being generated by their testbed. A good dataset should include both attack and normal traffic.

Taking a different approach to building their testbed, Bhatia et al. [180] employed physical machines arranged appropriately in a local network. Their testbed was tasked with simulating flash events and various types of DDoS attacks, the latter through the use of specialized software called Botloader and IP-Aliasing. Compared to our virtualized approach, their choice of using physical machines incurs added costs (for the physical machines), is not easily deployable as mentioned by the research team itself [180] and does not include the added resiliency that virtualized environments offer. Additionally, our approach included a greater variety of Botnet activities, including but not limited to DDoS, DoS and Port Scanning.

Similarly to [180], Behal et al. [181], developed their testbed, DDoSTB to generate DDoS attacks and flash events. Their approach was to use a combination of real computers arranged locally into subnetworks, and emulation software that created multiple virtual nodes per physical. As mentioned for [180], the use of physical machines in such a testbed lacks the resiliency, ease and speed with which Virtual Machines can be deployed. Also, again the focus of the testbed was DDoS attacks, which, although a significant and particularly destructive capability of Botnets is far from the only kind of action that they can perform, as has been observed, that Botnets in the wild exhibit a number of diverse malicious actions, such as Keylogging, Phishing, Data theft, DDoS, DoS and Malware proliferation.

Comparable to [180, 181] but more elaborate in their implementation, Sharafaldin et al. [59] crafted a testbed relying on physical machines, which were separated into two networks, the Victim-network and the Attack-network. Their approach made use of several popular Operating Systems, such as Windows (7Pro, 8.1, Vista), Ubuntu (16.4,14.4, Server) and Kali Linux, a choice that mirrors our own. Moustafa et al. [57] relied on the IXIA Perfect Storm in their testbed implementation to generate both normal and malicious network traffic for the purpose of generating the UNSW-NB15 dataset. Through IXIA, the researchers installed three virtual servers, two of which were tasked with generating normal traffic with the third performing attacks on them.

Doshi et al. [33] work focused on generating an ML model that would identify IoT-produced DDoS attacks. In their approach, they made use of physical IoT consumer appliances, which were deployed in a local network. Normal traffic was collected by interacting with the IoT devices, while DoS attacks were simulated. The main differences between their proposed testbed and the one we implemented, is scale and attack variety. For our testbed, we chose to use four Kali Linux machines to simulate both DoS and DDoS attacks, along with other Botnet actions.

The main novelty of the proposed dataset, is the introduction of the IoT element in the environment. Namely, we employed popular middleware (Node-red), to simulate the existence of IoT devices in our Virtual Network. This IoT simulated traffic is in the form of MQTT, a publish-subscribe communication protocol implemented over TCP/IP, often used for lightweight network communication, such as IoT [182]. In contrast, IoT devices were not represented in the testbeds that were presented in the previous section [32, 57, 59, 179, 180, 181] with the exception of [33].

Choosing this virtualized setup carries a number of pros and cons. To start with the pros, using a virtualized environment means that the setup is portable and easy to set up with a relatively low cost. Additionally, using simulations to represent the servers, PCs and IoT devices made the launching of malicious attacks easier and safer, with the extra bonus that the machines could be recovered easily if necessary. From a security perspective, not using actual Botnet malware to infect our machines made the process a lot safer, as by doing so we ran the risk of either unwillingly taking part in real attacks (our machines become part of a real botnet).

Furthermore, many newer versions of Bot malware can detect a virtualized environment, producing fake data as subterfuge. With regards to the generation of realistic normal network traffic, the Ostinato [62] software was used, to make the dataset appear as if it were collected from a real-world network. Lastly, we executed a set of standard Botnet attacks, which makes this dataset useful for the development of realistic Botnet traffic detection.

On the other hand, using a virtualized environment prevents us from launching in-depth attacks against an IoT's firmware and hardware, thus somewhat limiting the depicted ways through which an attack against such machines can be launched. Nevertheless, the collected dataset is adequate for our purposes. In addition, as an expansion, an even more diverse group of attacks could be performed, such as application layer DoS/DDoS, Layer 1,2 attacks on physical IoT devices, something which requires the use of real IoT devices, access to which we did not have at the time of the experiments.

3.2.3. Existing network datasets and their forensics analytics limitations

Since the applications of forensics discussed in the IoT and Forensic analytics subsection employ machine learning techniques, they require big datasets for analyzing network flows, differentiating between normal and abnormal traffic, and producing forensic reports, which could be useful to forensic specialists in law enforcement. The development of a realistic network dataset is a very important task for designing network forensics, intrusion detection and privacy-preserving models [183]. Over the years, several datasets have been produced [59] and although a good number of them remain private, due to primarily privacy concerns, some have become publicly available. The most commonly used datasets are briefly explained below, with a comparison between them and Bot-IoT given in Table 3.2.

Table 3.2: Comparison of datasets (T=true, F=false)

Dataset	Realistic testbed configuration	Realistic traffic	Labeled data	IoT traces	Diverse attack scenarios	Full packet capture	New generated features
Darpa98	T	F	T	F	T	T	F
KDD99	T	F	T	F	T	T	T
DEFCON-8	F	F	F	F	T	T	F
UNIBS	T	T	T	F	F	T	F
CAIDA	T	T	F	F	F	F	F
LBNL	F	T	F	F	T	F	F
UNSW-NB15	T	T	T	F	T	T	T
ISCX	T	T	T	F	T	T	T
CICIDS 2017	T	T	T	F	T	T	T
TUIDS	T	T	T	F	T	T	T
Bot-IoT	T	T	T	T	T	T	T

- The DARPA 98 dataset was generated by MIT'S Lincoln Lab for assessing intrusion detection systems. The resulting dataset was produced in a period of 7 weeks, was made up of 4GB of binary data and simulated a small Air Force network connected to the Internet [184][185], which was later enhanced in 1999 to generate the features in the KDD99 dataset.

- The KDD99 dataset was generated from the DARPA 98 dataset for evaluating intrusion detection systems that distinguish between inbound attacks and normal connections [58][55][185]. Even though it is still used to this day, it has several problems, for example, non-normal distributions of attack and normal data named the imbalanced learning problem. The NSL-KDD dataset was proposed to address the limitations of the KDD99, but the two versions are outdated and do not reflect current normal and attack events [55][59].

- The DEFCON-8 dataset consists of port scanning and buffer overflow attacks, which were recorded during a "Capture the Flag" competition [59]. As it lacks a significant quantity of normal background traffic, its applicability for evaluating Network Forensics and IDS systems is limited.

- The UNIBS [56][186] dataset was developed by the University of Brescia, Italy. In their configuration, the researchers installed 20 workstations running the Ground Truth daemon and traffic was collected through tcpdump at the router to which they were collected. Although the researchers used a realistic configuration, there are some drawbacks to this dataset. First, the attack scenarios are limited to DoS attacks. Secondly, the dataset exists in packet form with no extra features generated on it. Additionally, no information is given about the labels.

- The CAIDA datasets [186][187] are collections of varied data types. They are comprised of anonymized header traffic, excluding the payload. The datasets are made up of very specific attacks, such as DDoS. One popular dataset from the CAID collection is the CAIDA DDoS 2007, which includes one hour of anonymized attack traces from DDoS attacks that took place in August 2007. One drawback of the CAIDA datasets is that they did not have a ground truth about the attack instances. Additionally, the gathered data was not processed to generate new, features which could improve the differentiation between attack and normal traffic.

- The LBNL [188][186] dataset consists of anonymized traffic, which consists of only header data. It was developed at the Lawrence Berkley National Laboratory, by collecting real inbound, outbound and routing traffic from two edge routers. Similarly to the UNIBS, the labelling process is lacking and no extra features were generated, with the data existing as a collection of .pcap files.

- The UNSW-NB15 is a dataset developed at UNSW Canberra by Mustafa et al. [57]. The researchers employed IXIA perfect storm to generate a mixture of benign and attack traffic, resulting in a 100GB dataset in the form of PCAP files with a significant number of novel features generated. The purpose of the produced dataset was to be used for the generation and validation of intrusion detection. However, the dataset was designed based on a synthetic environment for generating attack activities.

- The ISCX dataset [189][190] was produced at the Canadian Institute for Cybersecurity. The concept of profiles was used to define attack and distribution techniques in a network environment. Several real traces were analyzed to generate accurate profiles of attacks and other activities to evaluate intrusion detection systems. Recently, a new dataset was generated at the same institution, the CICDS2017.

- The CICIDS2017 [59] is comprised of a variety of attack scenarios, with realistic user-related background traffic generated by using the B-Profile system.

Nevertheless, the ground truth of the datasets, which would enhance the reliability of the labelling process, was not published. Furthermore, applying the concept of profiling, which was used to generate these datasets, in real networks could be problematic due to their innate complexity.

- The TUIDS [46][186] dataset was generated by the Tezpur University, India. This dataset features DDoS, DoS and Scan/Probing attack scenarios, carried out in a physical testbed. However, the flow level data do not include any new features other than the ones generated by the flow-capturing process.

Although various research studies have been conducted [46, 56, 57, 58, 59, 187, 188, 189] to generate network datasets, the development of realistic IoT and network traffic dataset that includes recent Botnet scenarios still is an unexplored topic. More importantly, some datasets lack the inclusion of IoT-generated traffic, while others neglected to generate any new features. In some cases, the testbed used was not realistic while in other cases, the attack scenarios were not diverse. This chapter seeks to address the shortcomings by designing the new Bot-IoT dataset and evaluate it using multiple forensics mechanisms, based on machine and deep learning algorithms.

3.3. The proposed Bot-IoT dataset

3.3.1. Overview of proposed testbed

The proposed testbed consists of three components, namely: network platforms, simulated IoT services, and extracting features and Forensics analytics. First, the network platforms include normal and attacking virtual machines (VMs) with additional network devices such as a firewall and tap. Second, the simulated IoT services, which contain some IoT services such as a weather station. These are simulated through the Node-red tool [61]. Third, extracting features and forensics analytics, where the Argus tool [191] was used in order to extract data features, and afterwards statistical models and machine learning techniques were employed in order to assess the feature vectors for discriminating normal and abnormal instances. More details of the components are discussed below.

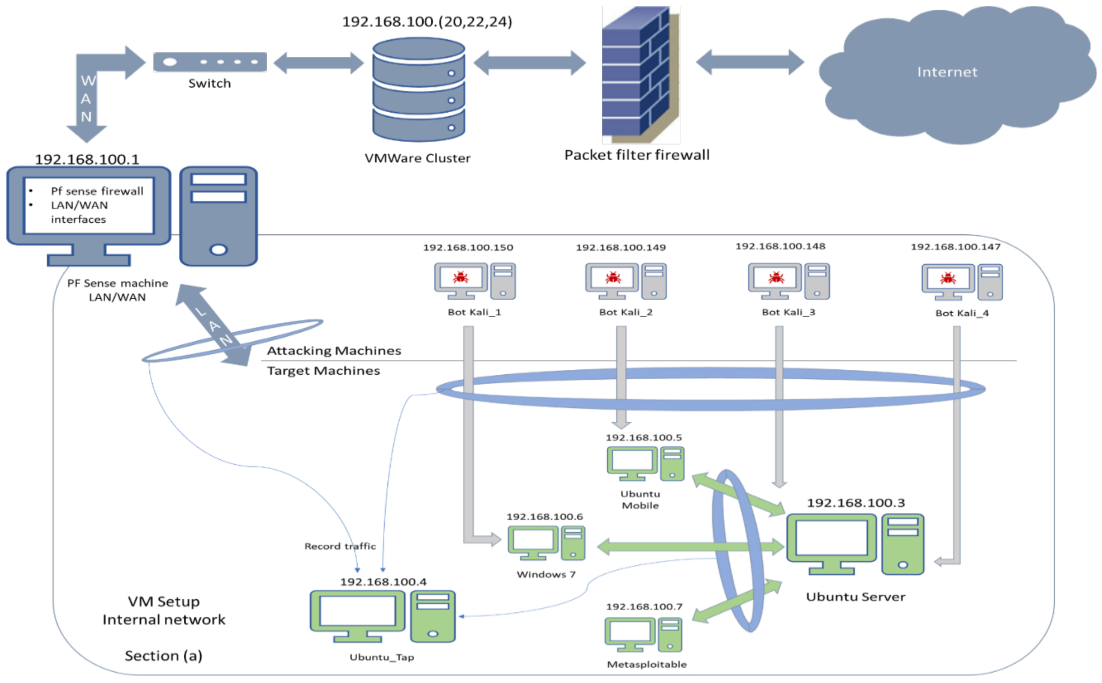


Fig. 3.1: Testbed environment of the new Bot-IoT dataset

3.4. Network platforms

We designed the testbed environment at the Research Cyber Range lab of UNSW Canberra. Virtual Machines that were prepared for this task, were ported into an ESXi [192] configured cluster and managed through a vSphere platform [193]. In Figure 3.1, we depict the testbed of the Bot-IoT dataset, where several VMs are connected to LAN and WAN interfaces in the cluster and are linked to the Internet through the PFSense machine. On the Ubuntu VM platforms, the Node-red tool was used for simulating various IoT sensors which were connected with the public IoT hub, AWS [194]. We developed Java scripts on the Node-red tool for subscribing and publishing IoT services to the IoT gateway of the AWS via the Message Queuing Telemetry Transport (MQTT) protocol [182], as detailed in the following sub-section.

There were also a packet filtering firewall and two Network Interface Cards (NICs) configured in the environment. One of the NIC was configured for LAN and the other one for WAN. The main reason for using this firewall is to ensure the validity of the dataset labelling process, as it enables to manage network access by monitoring incoming and outgoing network packets, based on specific source and destination internet protocol (IP) addresses of the attack and normal platforms. VMs which needed to communicate with the Internet sent their traffic through the PFSense machine which in turn, forwarded the traffic through a switch and a second firewall before it could be routed further into the Internet.

Our network of VMs consists of four Kali machines, Bot Kali_1, Bot Kali_2, Bot Kali_3, and Bot Kali_4, an Ubuntu Server, Ubuntu mobile, Windows 7, Metasploitable and an Ubuntu Tap machine. The Kali VMs, which belong to the attacking machines, performed port scanning, DDoS and other Botnet-related attacks by targeting the Ubuntu Server, Ubuntu mobile, Windows 7 and Metasploitable VMs. In the Ubuntu Server, a number of services had been deployed, such as DNS, email, FTP, HTTP, and SSH servers, along with simulated IoT services, in order to mimic real network systems.

To generate a massive amount of normal traffic, we used the Ostinato tool [62], due to its flexibility of generating realistic benign traffic with given IPs and ports. We also maintained periodically normal connections between the VMs by executing normal functions of the services installed on the Ubuntu server, such examples include the DNS server, which resolved the names of the VMS to their IPs and the FTP server, used to transfer particular files between the VMs. To collect the entire normal and attack raw packet volume exchanged within the configured network, the T-shark tool was used on the Ubuntu Tap machine, by setting its NIC in a promiscuous mode that ensured the scalability of the testbed.

3.4.1. Simulated IoT services

In order to simulate the network behaviour of IoT devices, we employed the Node-red tool [61]. Node-red is a popular middleware used to connect IoT physical devices with their backend cloud server and applications, improving and speeding up communications between the various parts of an IoT deployment. On the Node-Red tool, we developed JavaScript code that mimicked IoT sensors such as temperature, pressure and humidity sensors.

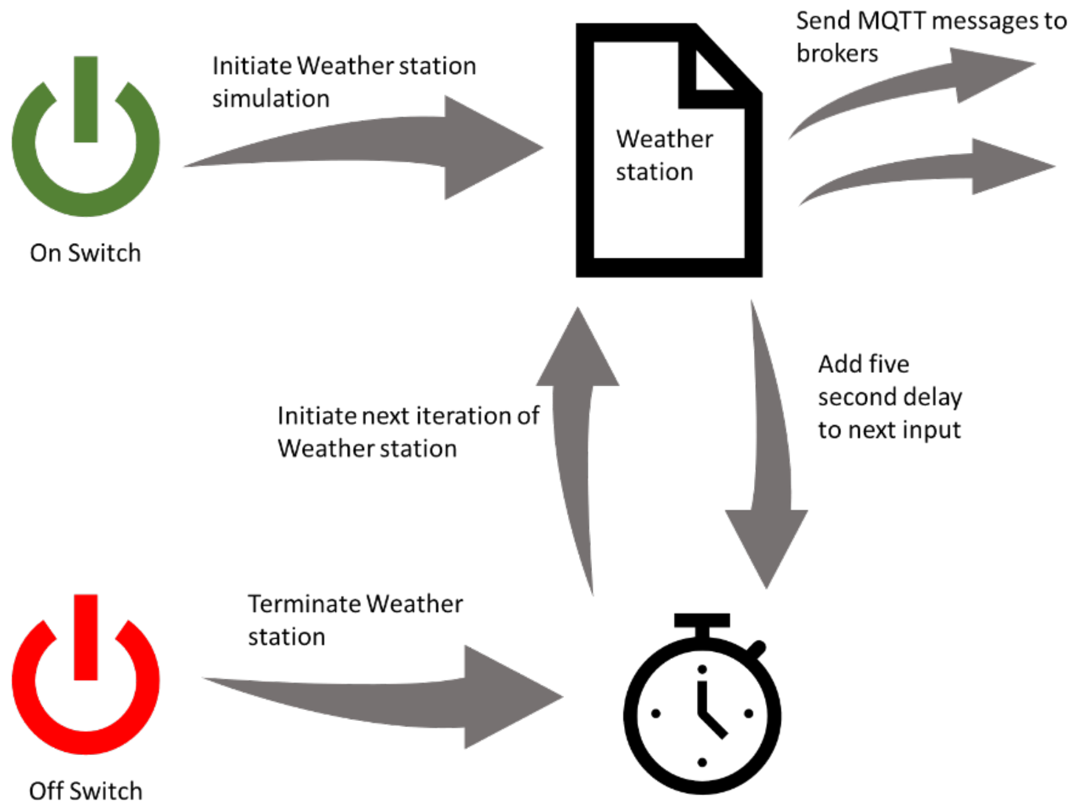

Initiate Weather station simulation
On Switch
Weather station
Send MQTT messages to brokers
Add five second delay to next input
Initiate next iteration of Weather station
Terminate Weather station
Off Switch

Fig. 3.2: Flowchart of weather IoT simulation of the node-red tool used in the dataset

The various pieces of code were triggered for publishing and subscribing to the topics, as shown in the example of Figure 3.2. The MQTT protocol [182] was used as a lightweight communication protocol that links machine-to-machine (M2M) communications, making it a viable choice for IoT solutions. It works in a publish/subscribe model, where a device publishes data to an MQTT broker (server-side) under a topic, which is used to organize the published data and allows clients to connect to the broker and fetch information from the topic of the device they wish to interact with.

We applied the following IoT scenarios in the testbed of the dataset:

1. A weather station (Topic:*/smarthome/weatherStation*), which generates information on air pressure, humidity and temperature.
2. A smart fridge (Topic:*/smarthome/fridge*), which measures the fridge's temperature and when necessary adjusts it below a threshold.

Algorithm 3.1 Weather Station

```
var temp, hum,pres,id;
var timestamp=(new Date().toGMTString());
const maxTemp=40;//Celcius
const minTemp=-5;
const initPres=1.013;//Bars
temp=msg.payload["temperature"]||Math.ceil(Math.random()*(maxTemp-
minTemp))+minTemp;
pres=msg.payload["pressure"]||initPres;
hum=msg.payload["humidity"]||Math.random()*100;
if(Math.random()>=0.05){//Has 0.05 probability to change temperature OR if temperature is
above 6.
temp=temp+(0.5*Math.random()*(temp-1<minTemp?(1):(temp+1>maxTemp?(-
1):(Math.random()>0.5?(1):(-1)))));}
if(!isNaN(msg.payload)){
id=msg.payload;}
else{
id=msg.payload["id"];
pres=pres+((((Math.random()*8)%8)<1)?(Math.random()*(0.5+0.5)-0.5):(0));//0.125 chance
to change pressure value by an increment in [-0.5,0.5]
hum=hum+((((Math.random()*6)%6)<1)?(hum-5<0?(Math.random()*(-
5)+5):(hum+5>100?(Math.random()*(5)-5):(Math.random()*(5+5)-5))):(0));}
msg.payload={"id":id,"timestamp":timestamp,"temperature":temp,"pressure":pres,"humidity":hum}
return msg;
```

Algorithm 3.2 Fridge

```
var FridgeTemp,id;
var timestamp=(new Date().toGMTString());
var tempMsg="Temperature low";
FridgeTemp=msg.payload["Fridge Temperature"] || Math.ceil(Math.random()*(14));
if(Math.random()>=0.05||FridgeTemp>=6){//Has 0.05 probability to change temper-
ature OR if temperature is above 6.
FridgeTemp=FridgeTemp+(Math.random().toFixed(1)*(1.5)*(FridgeTemp>=6?(-
1):(1)));}
if(FridgeTemp>=6){
tempMsg="Temperature high";}
if(!isNaN(msg.payload)){
id=msg.payload;}
else{ id=msg.payload["id"];}
msg.payload={"id":id,"Timestamp":timestamp,"Fridge                    Tempera-
ture":FridgeTemp,"Condition":tempMsg}
return msg;
```

3. Motion-activated lights (Topic:*/smarthome/motionLights*), which turn on or off based on a pseudo-random generated signal.

Algorithm 3.3 Motion activated light

```
var id, MotionDetected, lightsOn;
var timestamp=(new Date().toGMTString());
var message="Lights off.";
const initLightCondition=false;
const initMotionSensor=false;
var chance=Math.random()*5%5;
if(!msg.payload["Motion Detected"]){//if previously motion wasn't detected
MotionDetected=(chance>2)?(MotionDetected=true):(MotionDetected=false);//0.2
(20%) chance motion will be detected.
if(MotionDetected){
lightsOn=true;}
else{
lightsOn=false;}
else{
MotionDetected=lightsOn=false;}
if(!isNaN(msg.payload)){
id=msg.payload;}
else{
id=msg.payload["id"];}
message=(lightsOn?("Lights on."):("Lights off."));
msg.payload={"id":id,"timestamp":timestamp,"Motion De-
tected":MotionDetected,"Lights Condition":lightsOn,"message":message}
return msg;
```

4. A remotely activated garage door (Topic:*/smarthome/garageDoor*), which opens or closes, based on a probabilistic input.

5. A smart thermostat (Topic:*/smarthome/thermostat*), which regulates the house's temperature by starting the Air-conditioning system.

The five IoT scenarios were connected to both the Ubuntu server, where the Mosquitto MQTT broker [195] was installed, as-well-as the AWS IoT hub. While running the testbed environment, MQTT messages were published periodically from all clients to both brokers. The connections allowed us to simulate regular IoT traffic since the MQTT brokers were used as intermediaries that connected smart devices with web/smartphone applications.

Algorithm 3.4 Garage Door

```
var id, smartphoneSignal, doorState;
var timestamp=(new Date().toGMTString());
const initSmartphoneSignal=false;
const deviceTag="Garage Door";
const DoorOff=false;
const DoorOpen=true;
doorState=msg.payload["door state"]||DoorOff;
smartphoneSignal=msg.payload["sphone signal"]||initSmartphoneSignal;
if(!isNaN(msg.payload)){
id=msg.payload;}
else{
id=msg.payload["id"];
smartphoneSignal=(Math.random()*10)<1?(true):(false);
doorState=(smartphoneSignal?true:false);}
msg.payload={"id":id,"device title":deviceTag ,"door state":doorState, "door
state text": (doorState?"Garage door open":"Garage door closed"),"sphone sig-
nal":smartphoneSignal}
return msg;
```

3.5. Extracting features and forensics analytics

After collecting the pcap files from the virtualized setup, with normal and attack traffic in the same files, we extracted the flow data, by using the Argus tools and produced .argus files. Following the flow extraction process, the produced flow data was imported into a MySQL database for further processing. We then employed statistical measures using Correlation Coefficient [66] and Entropy [67] techniques to assess the original dataset and select important feature, as described in Section 6. New features were generated based on the transactional flows of network connections in order to discover normal and intrusive events. Finally, three Machine Learning models, which could be applied for forensic purposes, were trained and validated on several versions of the dataset to assess the value of the dataset compared with other datasets, as discussed in Section 7.

3.5.1. Network Flow extraction process

Capturing network traffic while ensuring the labelling process is not an easy task, as the synchronization of sending and receiving packets and later tagging these packets either normal or attack should be timely and automatically developed. In order to accomplish this task, we developed some scripts on the Cron Linux

Algorithm 3.5 Thermostat

```
var id, currentTemperature, AC_State;
var timestamp=(new Date().toGMTString());
const deviceTag="Smart Thermostat";
const defaultTemperature=25;
const AC_Default=false;
const DoorOpen=true;
currentTemperature=msg.payload["current temperature"]||defaultTemperature;
AC_State=msg.payload["AC_state"]||AC_Default;
if(!isNaN(msg.payload)){
id=msg.payload;}
else{
id=msg.payload["id"];
if(currentTemperature!=25){
AC_State=true;}
else{
AC_State=false;}
if(AC_State){
if(currentTemperature-25<1&&currentTemperature-25>0){
currentTemperature=currentTemperature-(currentTemperature-25);
AC_State=false;}
else if(currentTemperature-25>-1&&currentTemperature-25<0){
currentTemperature=currentTemperature-(currentTemperature-25);
AC_State=false;}
else{
currentTemperature=currentTemperature+((currentTemperature>25)?(-
1):(+1))*(Math.random())*0.5;}
}
else{
currentTemperature=currentTemperature+(((Math.random*10>8)?(-
1):(+1))*(Math.random()*10));}
}
msg.payload={"id":id,"device title":deviceTag ,"current tempera-
ture":currentTemperature,"AC_state":AC_State, "state of thermostat": "House
temperature "+(currentTemperature)+ (AC_State?" AC on.":" AC off.")};
return msg;
```

functions [196] over the Ubuntu Tap VM. When the scripts ran on a given time, a particular normal or attack scenario had to be executed. For example, during the generation of DDoS, we scheduled the execution of custom bash scripts which invoked hping3 and golden-eye to run the DDoS attacks, while simultaneously normal traffic was generated in the background. At the same time, the T-shark tool [65] was running, to capture raw packets and store them in 1 GByte pcap files to ease extracting network features.

We scheduled different types of attacks to run at different times, with normal background traffic being constantly generated. By doing so, we ensured that different types of attacks would be performed at different times allowing us to organize the produced pcap files, based on attack category and subcategory. For this purpose, the attacking Kali Bots 1-4 and the recording Ubuntu Tap, had to be synchronized, so that Ubuntu Tap could halt the recording of a particular attack's pcap files and start the next one scheduled. The normal traffic, which was mixed with the attack traffic was generated by the Ostinato [62] program that ran in the Ubuntu_Server VM. Knowing the IP addresses of both attacker and victim machines, enabled us to differentiate between normal and attack traffic, as we ensured that between the two groups, only attacking traffic would be transferred.

After collecting the pcap files, the Argus tool [191] was used to generate the relevant network flows. The pcap files were converted into Argus format by using the Argus client program. Then, the *rasqlinsert* command was applied to extract network flow information, and simultaneously log the extracted features into MySQL tables. The final features produced by Argus during the Network flow extraction process are listed in Table 3.3.

Additionally, the group "saddr", "sport", "daddr", "dport", "proto" are considered network flow identifiers, as this information is capable of uniquely identifying a flow at any given time and assisting in the labelling process. To label the data for use with machine learning processes, we employed "alter table" queries to introduce the new columns and "update" queries to modify the values based on saddr and daddr values. In the dataset, attack instances are labelled with "1" and normal ones are labelled with "0" for training and validating machine learning models through a binary classification. In addition to that, we have further introduced attack category and subcategory attributes, which could be used for training and validating multiclass classification models.

3.5.2. New feature generation

We developed new features that were generated based on the features listed in Table 3.3. The main purpose of this process is to improve the predictive capabilities of classifiers. The new features demonstrated in Table 3.4 were designed over a sliding window of 100 connections. The number 100 although chosen arbitrarily, pays a significant role in the generation of these new features, as it captures the statistics of groups of flows, in a relatively small time-window, inside of which,

Table 3.3: Features and descriptions

Feature	Description
pkSeqID	Row Identifier
Stime	Record start time
flgs	Flow state flags seen in transactions
flgs_number	Numerical representation of feature flags
Proto	Textual representation of transaction protocols present in network flow
proto_number	Numerical representation of feature proto
saddr	Source IP address
sport	Source port number
daddr	Destination IP address
dport	Destination port number
pkts	Total count of packets in transaction
bytes	Totan number of bytes in transaction
state	Transaction state
state_number	Numerical representation of feature state
ltime	Record last time
seq	Argus sequence number
dur	Record total duration
mean	Average duration of aggregated records
stddev	Standard deviation of aggregated records
sum	Total duration of aggregated records
min	Minimum duration of aggregated records
max	Maximum duration of aggregated records
spkts	Source-to-destination packet count
dpkts	Destination-to-source packet count
sbytes	Source-to-destination byte count
dbytes	Destination-to-source byte count
rate	Total packets per second in transaction
srate	Source-to-destination packets per second
drate	Destination-to-source packets per second
attack	Class label: 0 for Normal traffic, 1 for Attack Traffic
category	Traffic category
subcategory	Traffic subcategory

patterns of several attacks can be discovered. In order to generate these features in MySQL DB, we made use of stored procedures.

Table 3.4: Generated flow features

	Feature	Description
1	TnBPSrcIP	Total Number of bytes per source IP.
2	TnBPDstIP	Total Number of bytes per Destination IP.
3	TnP_PSrcIP	Total Number of packets per source IP.
4	TnP_PDstIP	Total Number of packets per Destination IP.
5	TnP_PerProto	Total Number of packets per protocol.
6	TnP_Per_Dport	Total Number of packets per dport.
7	AR_P_Proto_P_SrcIP	Average rate per protocol per Source IP. (calculated by pkts/dur)
8	AR_P_Proto_P_DstIP	Average rate per protocol per Destination IP.
9	N_IN_Conn_P_SrcIP	Number of inbound connections per source IP.
10	N_IN_Conn_P_DstIP	Number of inbound connections per destination IP.
11	AR_P_Proto_P_Sport	Average rate per protocol per sport.
12	AR_P_Proto_P_Dport	Average rate per protocol per dport.
13	Pkts_P_State_P _Protocol_P_DestIP	Number of packets grouped by state of flows and protocols per destination IP.
14	Pkts_P_State_P _Protocol_P_SrcIP	Number of packets grouped by state of flows and protocols per source IP.

3.6. Benign and Botnet scenarios

3.6.1. Benign Scenarios

In the testbed environment, we design a typical smart-home configuration. Initially, five smart devices were simulated and operated locally. We employed the Node-Red toot to connect smart devices and the corresponding Cloud infrastructure for generating normal/benign network traffic. Moreover, the Ostinato tool was also utilized to generate a huge amount of normal traffic between the VMs, like network production systems.

The configuration of the VMs and utilized platforms represents a realistic smart-home network, as the five IoT devices: 1) Smart Refrigerator, 2) Smart Garage door, 3) Weather Monitoring System, 4) Smart Lights, and 5) Smart thermostat, could be deployed in smart homes. Also, the generated messages were transferred to a Cloud Service provider (AWS) using the MQTT protocol. The statistics of normal traffic included in the dataset are shown in Table 3.5.

Table 3.5: Statistics of normal instances included in Bot-IoT dataset

Protocol	Number
UDP	7225
TCP	1750
ARP	468
IPV6-ICMP	88
ICMP	9
IGMP	2
RARP	1
Total	9543

3.6.2. Botnet scenarios

As previously mentioned, we used four Kali Linux VMs to launch cyber-attacks in parallel for implementing different botnet scenarios, as depicted in Figure 3.1. The cyber-attacks and their tools considered in the Bot-IoT dataset are described as follows:

- **Probing attacks** [197][198][199][200]- are malicious activities that gather information about victims through scanning remote systems, also so-called, fingerprinting [197][199]. The probing types included in the dataset are further discussed below.

  Probing can be split into subcategories, first based on actions performed during probing and second based on the information gathering target. First, according to the actions performed, probing can be split into passive and active probing [198][199]. During passive probing, an attacker simply captures any and all available packets in the network, thus operating in a stealthy manner [198][199]. On the other hand, during an active probe, an attacker generates network traffic, targeting the system, and records the responses, comparing them with known responses which allow them to make inferences about services and OS [198][199]. With regards to the goal of the probe, there are two major subcategories, OS and service fingerprinting. In OS fingerprinting, a scanner gathers information about the remote system's OS by comparing its responses to pre-existing ones or based on differences in TCP/IP stack implementations. In service fingerprinting (scanning), a scanner identifies the services which run behind the system's ports (0-65535), by sending request packets [199]. Here, we will be using active scans, as passive scans produce close to zero amount of generated traffic.

- Port scanning: we used the Nmap and Hping3 tools in order to perform a number of different types of port scans. The Nmap [200] was launched to scan open services and protocols of the VMs, for example, *nmap -sT 192.168.100.3*, where *sT* issues a complete TCP connection and *192.168.100.3* is the IP of the Ubuntu server. Likewise, Hping3 [64] was employed to perform similar port scans with an example being, *hping3 -S -scan 1-1000 192.168.100.3*, where *S* issues a SYN scan and *scan* dictates the port numbers to be scanned.
- OS fingerprinting: we used the Nmap and Xprobe2 tools to launch different types of OS fingerprint scans. The Nmap [200] tool was used to identify the OS of our target VMs with different options, for example, *nmap -sV -T5 -PO -O 192.168.100.3*, where *sV* specifies a SYN scan, *T5* that the scan is as overt as possible, *PO* to include IP protocol ping packets and *O* to enable OS scanning. The Xprobe2 [201] tool was used in conjunction with Nmap. While performing our scans, we used the default operations of Xprobe2, with no options specified.

• **Denial of Service** [181][33][12][202]- are malicious activities that attempt to disrupt a service, thus making it unavailable to legitimate users. The DDoS and DoS attack types included in the dataset are described as follows: Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks are performed by a group of compromised machines called Bots and target a remote machine, usually a server [181][33][12]. The purpose of such attacks is the disruption of services accessible by legitimate users. These attacks can be classified, based on their attack methodology. Two such groups are volumetric and protocol-based DDoS/DoS attacks [202]. Volumetric attacks generate a great number of network traffic, which either forces the victim to process through these attack-generated requests or cause the machine to crash, thus making the provided service unavailable. Protocol-based attacks abuse the mechanics of Internet protocols, which cause CPU and memory resources to be depleted, thus render the targeted machine unable to respond to requests. In our attack scenarios, we performed both DDoS and DoS and used the following protocols: TCP, UDP and HTTP.

- DDoS, DoS: We used the tool Hping3 [64] for both DDoS, DoS for TCP and UDP, for example, *hping3 –syn –flood -d 100 -p 80 192.168.100.3* where *syn* indicates a SYN TCP attack, *flood* indicates packets are sent as fast as possible, *d* specifies packet body size, *p* sets the targeted port. For HTTP DDoS and DoS attacks, we used the Golden-eye tool, one example being *goldeneye.py http://192.168.100.3:80 -m post -s 75 -w 1*, with *http://192.168.100.3:80* indicating the IP address of Ubuntu Server and the targeting Port number, *m* setting the method as post, *s* setting number of sockets, *w* specifying number of concurrent workers.

- **Information Theft** [203][204]- is a group of attacks where an adversary seeks to compromise the security of a machine in order to obtain sensitive data. The information theft attack types included in the dataset are described as follows:

  Information theft attacks can be split into subcategories, based on the target of the attack. The first subcategory is data theft. During data theft attacks, an adversary targets a remote machine and attempts to compromise it, thus gaining unauthorized access to data, which can be downloaded to the remote attacking machine. The second subcategory is keylogging. In keylogging activities, an adversary compromises a remote host in order to record a user's keystrokes, potentially stealing sensitive credentials. Attackers usually employ Advanced Persistent Threat (APT) methodology in conjunction with information Theft attacks, in order to maximize their attack's efficiency [203].

  – Data theft: we used the Metasploit framework [63] to exploit weaknesses in the target machines. For Windows 7 we exploited the SMB eternal blue vulnerability, while for Ubuntu Server we took advantage of weak administrator credentials. Post exploitation, we set-up a reverse meterpeter TCP connection through which exfiltration of entire directories became possible.

  – Keylogging: we used the Metasploit framework [63] to exploit the same weaknesses we used during Data theft. For Windows 7, meterpeter provided adequate software to perform keylogging, although the unpredictable shutdowns of the exploit itself rendered a collection of keystrokes impossible. For Ubuntu Server, we used the logkeys [205] software to record keystrokes. Initially, a dictionary attack was launched through Hydra [206] on the SSH service. Then, through Metasploit, an ssh connection was established which was later upgraded to a sudo meterpeter

Table 3.6: Statistics of attacks in IoT-Bot dataset

Information gathering	Service scanning		nmap, hping3	1463364
	OS Fingerprinting		nmap, xprobe2	358275
Denial of Service	DDoS	TCP	hping3	19547603
		UDP	hping3	18965106
		HTTP	golden-eye	19771
	DoS	TCP	hping3	12315997
		UDP	hping3	20659491
		HTTP	golden-eye	29706
Information theft	Keylogging		Metasploit	1469
	Data theft		Metasploit	118
Total				73360900

connection, allowing us to access the logkeys software, record keystrokes in the compromised host and then download the recordings.

The statistics of attacks involved in the dataset are described in Table 3.6.

3.7. Statistics and machine learning methods

This section describes the theoretical background of statistical measures for evaluating optimal features and machine learning for forensically identifying cyber-attacks.

3.7.1. Statistical analysis techniques

- *Pearson Correlation Coefficient:* is used for measuring the linear relationship between the feature set of the Bot-IoT dataset. Its output ranges between [-1,1], and its magnitude indicates the strength of correlation between two feature vectors, and its sign indicates the type of correlation either positive or negative [66].
- *Entropy:* depicts the uncertainty or disorder between features [207] of the Bot-IoT dataset. By definition, high values of entropy equate to high uncertainty.

$$-\sum_x \sum_y (P(x,y) * \log P(x,y)) \tag{3.7.1}$$

Based on Equation 3.7.1, the produced entropy values would be greater than or equal to zero H(X,Y)$\geq$0, as such the minimum value of entropy is 0. We calculated the pairwise Shannon Joint Entropy [67] of all rows, excluding class features, resulting in an n,n table., where n is the number of features.

3.7.2. Machine and Deep Learning analysis techniques

Machine and Deep Learning models were used to evaluate the quality of Bot-IoT when used to train a classifier. The models that were trained were: Support Vector Machine (SVM), Recurrent Neural Network (RNN) and Long-Short Term Memory Recurrent Neural Network (LSTM-RNN).

- SVM: is based on the idea that data instances can be viewed as coordinates in an N-dimensional space, with N being the number of features. During training, a hyperplane is sought, that best separates the data into distinct groups (classes) and that maximizes the margin. In our work, we made use of an SVM classifier with a linear kernel [208].
- RNN: incorporates a form of memory in its structure [209]. The output of an RNN during an iteration depends both on the input at any given time, and the output of the hidden state of the previous iteration. What makes RNN stand out, is that contrary to other NNs, its output depends on both the current input as-well-as previous outputs, making it ideal for processing temporal data, such as the ones present in our dataset. Usual applications of RNNs include machine translation, speech recognition, generating Image descriptions.
- LSTM: is a special kind of Recurring Neural Network, where a collection of internal cells are tasked with maintaining a sort of memory which makes LSTMs ideal for finding temporally distant associations [210]. LSTM improve on RNN's " vanishing gradient" and "exploding gradient" problems, by incorporating a "memory cell" which handles updates in the model's memory. This improvement renders LSTMs ideal for learning long-term dependencies in data.

3.8. Experimental Results and Discussion

3.8.1. Pre-processing steps of Bot-IoT dataset

In order to extract the dataset from its MySQL tabular form, which contains the combined records from all the subsequent tables, where the labelling process took place, we introduced an auto-incrementing feature named “pkSeqID”, and then employed *"select * from IoT_Dataset_UNSW_2018 into outfile '/path/to/-file.csv'" Fields terminated by ',' Lines terminated by ' \n';*, to extract the dataset into CSV form. Doing so enabled us to easily process the data with various Python modules and also makes distributing the dataset easier, as CSV format is widely used for such purposes.

Furthermore, considering that the generated dataset is very large (more than 72.000.000 records and at 16.7 GB for CSV, with 69.3 GB pcap), it made handling the data very cumbersome. As such, we extracted 5% of the original dataset via the use of select MySQL queries similar to the ones mentioned previously. The extracted 5%, which we will refer to as the training and testing sets for the rest of the chapter, is comprised of 4 files of approximately 0.78 GB total size, and about 3 million records.

Additionally, it became necessary at some point of the evaluation process, to introduce discrete copies of numerical features. To do so, we grouped the numeric values into 5 bins of equal size, and later used the mathematical representation of the produced sets “(min, max)” of each bin as the new discrete value for each corresponding numeric in the dataset.

Moreover, due to the existence of certain protocols (ARP), source and destination port number values were missing (not applicable), as such, these values were set to -1, which is an invalid port number, again for the purpose of evaluation of the dataset.

We converted the categorical feature values in the dataset into consecutive numeric values for easily applying statistical methods. For example, the state attribute has some categorical values such as “RST”, “CON”, and “REQ” that were mapped into “1”, “2” and “3”.

Moreover, normalization was applied in order to scale the data into a specific range, such as [0,1], without changing the normality of data behaviours. This step helps statistical models and deep learning methods to converge and achieve their

Table 3.7: Machine Learning evaluation metrics

Accuracy	$ACC = \frac{TP}{TP+FP}$
Precision	$PPV = \frac{TP}{TP+FP}$
Recall	$TPR = \frac{TP}{TP+FN}$
Fall-out	$FPR = \frac{FP}{FP+TN}$

objectives by addressing local optimum challenges. We performed a Min-Max transformation on our data, according to the following formula:

$$x_i^{'} = (x_i - x_{min}) * \frac{(b-a)}{(x_{max} - x_{min})} + a \tag{3.8.1}$$

Where xmax and xmin are the initial max an min values from the original set, b and a are the new max and min set values and $x_i^{'} \in [a, b]$. For our purposes, a=0 and b=1, making the new set [0,1]. In order to measure the performance of the trained models, corresponding confusion matrices were generated, along with a collection of metrics, as given in Table 3.7.

3.8.2. Unsupervised Attribute evaluations

Initially, we followed a filter method feature selection. Such methods rely on statistical techniques to evaluate the quality of features rather than the optimization of ML models. The idea behind these evaluations is to identify any features that are highly correlated and reduce the dimensionality for improving the performances of machine learning models.

A. Correlation Coefficient

In order to calculate the correlation coefficient between the dataset's features, we developed a code in Python to rank the attribute strengths into a range of [-1, 1]. After calculating the Correlation Coefficient Matrix, we computed the average Correlation for each feature under scrutiny, thus gaining the "average correlation". The idea is that the features with the lowest Correlation Coefficient Average would introduce less ambiguity in our dataset. In Table 3.8, the features are represented in order from the lowest produced average correlation to the highest.

Table 3.8: Average Correlation Coefficient scores

Features	srate	drate	seq	rate	dur
Average CC	-0.00061	-0.00502	0.019941	-0.03508	0.051058
Features (1)	min	stddev	stime	ltime	flgs_number
Average CC (1)	0.070493	0.077707	0.102708	0.102716	-0.10681
Features (2)	state_number	mean	max	dpkts	dbytes
Average CC (2)	0.106846	0.170598	0.176318	0.237771	0.23879
Features (3)	sbytes	spkts	bytes	pkts	Sum
Average CC (3)	0.261992	0.265161	0.277309	0.284521	0.288727

Table 3.9: Average Joint Entropy scores for BoT-IoT features

Features	seq	mean	stddev	max	Min
Average CC	2.833223	2.465698	2.423786	2.088409	2.015226
Features (1)	state_number	flgs_number	stime	ltime	dur
Average CC (1)	1.963397	1.68267	0.798997	0.798997	0.665417
Features (2)	rate	sum	spkts	pkts	sbytes
Average CC (2)	0.664154	0.661829	0.661812	0.661783	0.661772
Features (3)	bytes	srate	drate	dbytes	dpkts
Average CC (3)	0.661768	0.661734	0.66172	0.661718	0.661718

B. Entropy

In order to calculate the joint entropy between our features, we generated Python code which traversed the loaded CSV files and calculated the subsequent sums of joint probability times the base 2 logarithm for that probability, as given in Equation 3.7.1. It was due to this measure that we performed the discretization that we mentioned at the beginning of this section.

Contrary to the Correlation Coefficient, Entropy depicts disorder in data, and as such, higher values indicate higher disorder (or randomness), which means that our features do not share much information and thus introduce less ambiguity in our dataset. Thus, we again produce a score value per feature, through calculating the average Joint Entropy, as depicted in the following table from higher to lower average Entropy.

C. Extraction of the 10 Best features

A direct comparison of both Entropy and Correlation scores is given in Figure 3.3. A feature will be considered ideal for our dataset if its Entropy score is large enough and its Correlation Score low enough. That would mean that that feature does not carry any redundant information that is shared with other features and that they are as unrelated to each other as possible. In order to compare the averages of these different statistical measures, score values were normalized in the range [0,1].

Considering that higher Correlation Coefficient values indicate highly correlated features, which is something we wanted to remove from our dataset, after performing the Min-Max transformation, we inverted the results $(1 - y_i)$, so as to bring the CC average scores in the same format as the Joint Entropy score (where higher values translate to higher randomness between features).

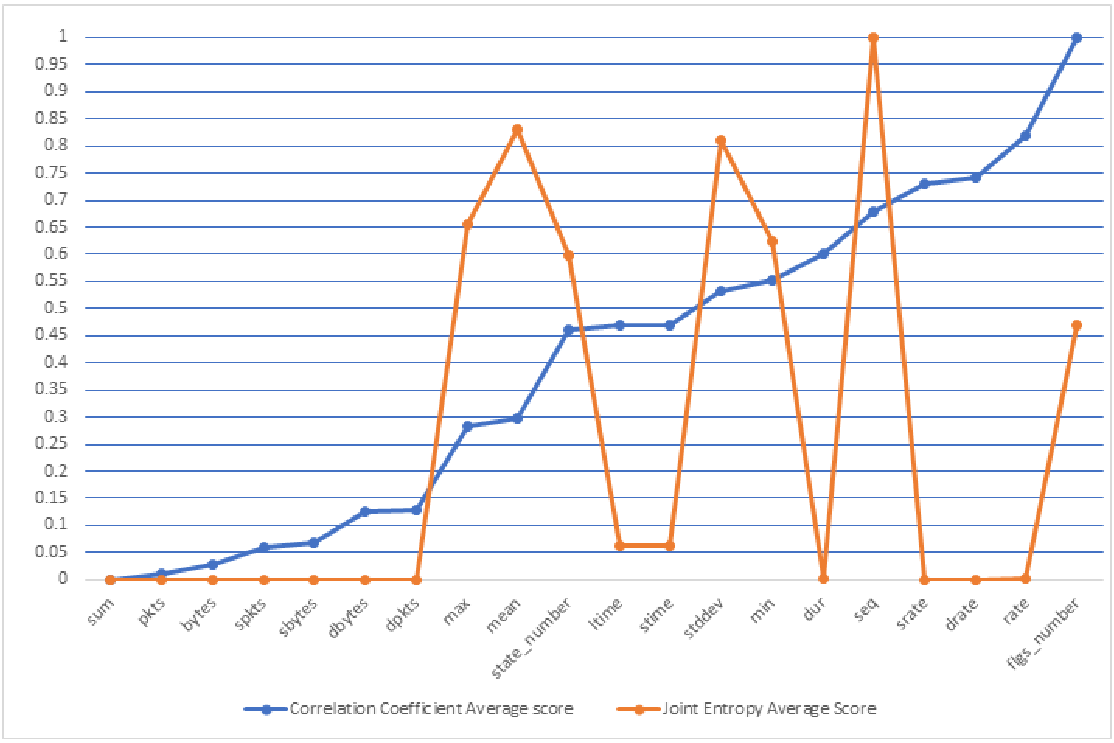


Fig. 3.3: Graph representation of features

We compared the new mapped values of Correlation Coefficient and Joint Entropy in order to extract a subset of 10 features which, overall, had the best scores in both statistical measures. As such, we identified toe following best 10 features: srate, drate, rate, max, state_number, mean, min, stddev, flgs_number, seq. Having completed the unsupervised evaluation process, we will further evaluate the worthiness of the final 10 features in the section “supervised evaluation”.

Table 3.10: Joint Entropy and Correlation Coefficnent average scores

Features	Average CC	Average JE
AR_P_Proto_P_Dport	0.071521	0.637848
AR_P_Proto_P_DstIP	0.034548	0.636399
AR_P_Proto_P_Sport	0.070876	0.636738
AR_P_Proto_P_SrcIP	0.034062	0.636084
drate	0.004333	0.635576
flgs_number	-0.13459	1.680341
max	0.18572	2.104321
mean	0.192966	2.490935
min	0.093812	2.022061
N_IN_Conn_P_DstIP	0.07164	1.430813
N_IN_Conn_P_SrcIP	0.077322	2.067795
Pkts_P_State_P_Protocol_P_SrcIP	0.263217	0.635667
Pkts_P_State_P_Protocol_P_DestIP	0.266336	0.635636
rate	0.074073	0.638024
seq	-0.02858	2.835564
srate	0.005537	0.63559
state_number	0.122872	1.973091
stddev	0.072967	2.440526
TnBPDstIP	0.168677	0.635674
TnBPSrcIP	0.161545	0.635689
TnP_PDstIP	0.271537	0.635701
TnP_Per_Dport	0.242596	0.63563
TnP_PerProto	0.130232	0.636924
TnP_PSrcIP	0.268818	0.635686

D. Secondary evaluation of features

In this section, we further evaluated the relationship between the independent features. For this stage, we employed similar tactics to the previous section of this chapter, that is, we calculated average scores based on the Pearson Correlation Matrix (Triangle) and Shannon Joint Entropy, for the 10 best features that were identified previously, combined with the 14 generated features, and then extracted the 10 best features from that group. Following are the Joined Entropy and Correlation Coefficient matrices and plots of their average scores.

We then mapped the average scores in the set [0,1] and plotted the values, in order to identify the 10 best features.

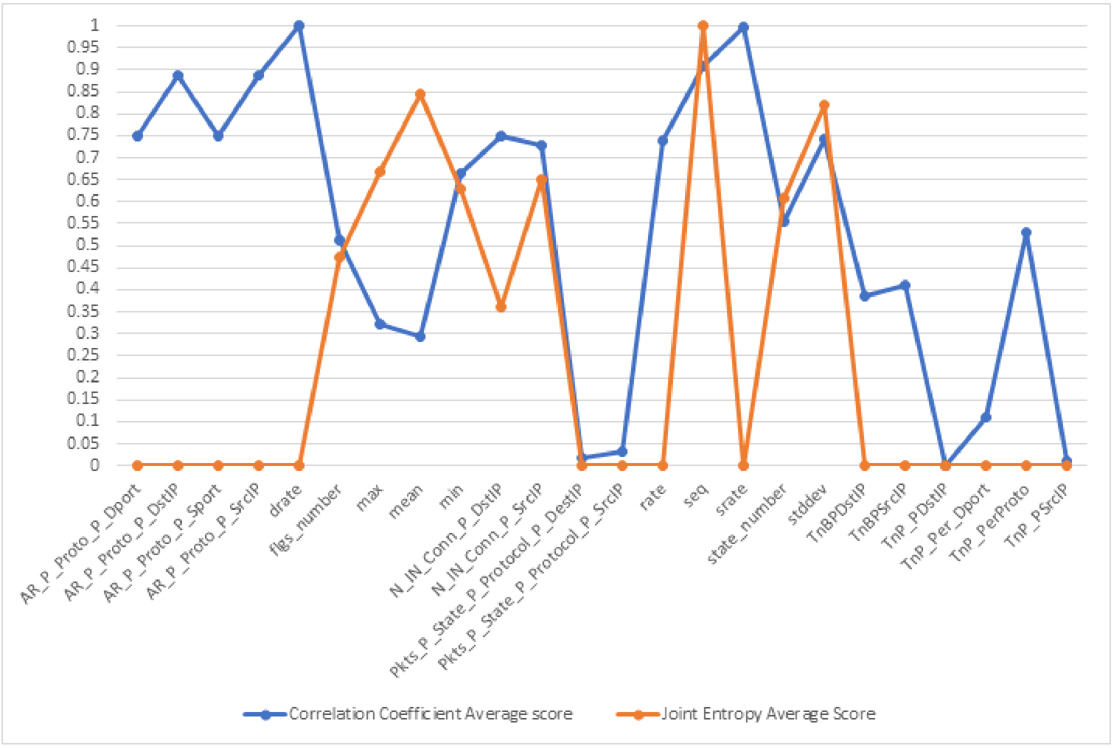


Fig. 3.4: Graph representation of features' scores

By observing Figure 3.4, we can determine that the 10 best features, that is, the 10 features with the highest combined average Correlation Coefficient and Joint Entropy are: *seq, stddev, N_IN_Conn_P_SrcIP, min, state_number, mean, N_IN_Conn_P_DstIP, drate, srate, max* .

3.8.3. Supervised evaluation

After the 10-best features were extracted, we employed supervised learning techniques to evaluate the quality of the dataset. Such methods rely on a model which is trained on the labelled data and then is capable of classifying new unlabeled instances.

A. Three classifiers for evaluations

Following the selection of the 10 best features, we applied three predictors on our data, and assessed their accuracy, in order to further eliminate any superfluous features. Specifically, the predictors we chose were a Support Vector Machine

(SVM), a Recurring Neural Network (RNN)and a Long Short-Term Memory RNN (LSTM-RNN).

A.1 SVM

The SVM model that was trained, was a linear Support Vector Classifier. The parameters of this classifier were penalty parameter (C=1), 4-fold cross-validation and a number of max iterations equal to 100000 on the final 10-best features. Similar settings were selected for the dataset version comprised of all available features, with the only difference being that max iterations were set to 1000.

The aforementioned setting was practically adjusted to measure the best performance of the SVM model. Initially, the SVM classifier was trained with default parameters, but it was later observed that by increasing the max iteration number, particularly for the second (all features included) model caused a longer training time. With regards to the number of folds, we observed a loss of accuracy when a higher number of folds was chosen.

Table 3.11: Confusion matrices of SVM models. (10-best feature model on the left, full-feature model on the right).

True\Predict	Normal (0)	Attack (1)	True\Predict	Normal (0)	Attack (1)
Normal (0)	477	0	Normal (0)	64	413
Attack (1)	426550	3241495	Attack (1)	0	3668045

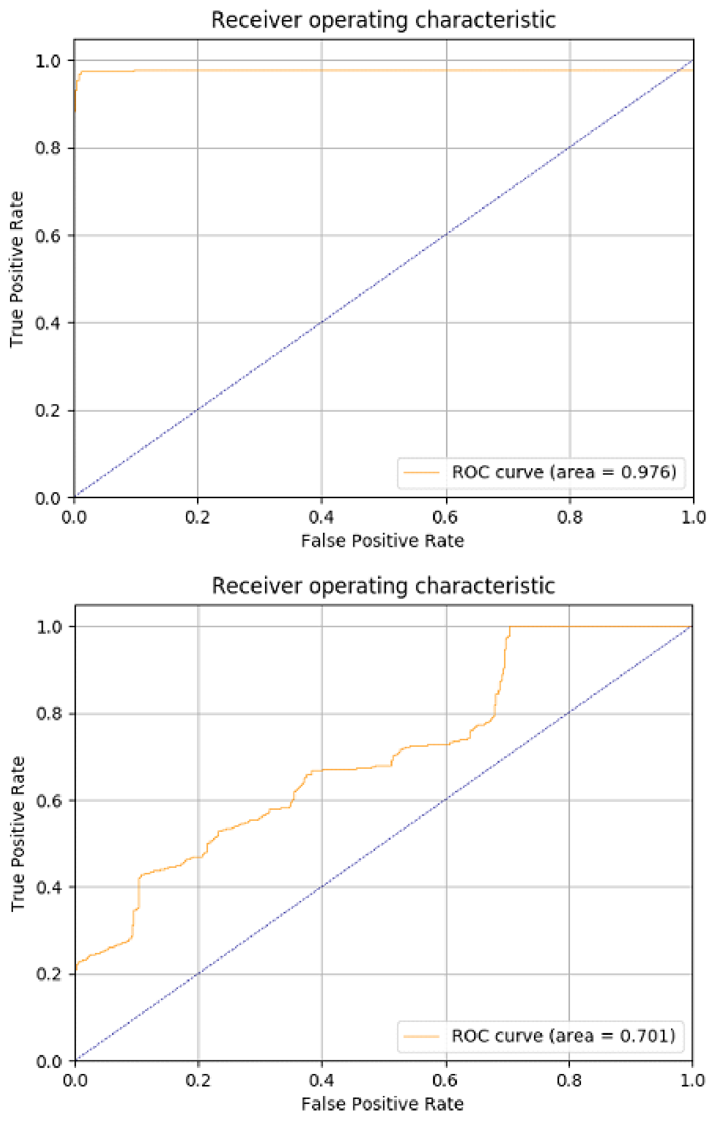


Fig. 3.5: ROC curve for SVM models.(10-best feature model on the left,full-feature model on the right).

A.2 LSTM

The LSTM models were defined to have 1 input layer with the number of neurons equal to the number of input features, two hidden layers and an output layer. For the 10-best features dataset, the Model was trained in 4 epochs with batch size 100. The neural network was comprised of 10 input neurons (in the first layer, the same number as the features), intermediate (hidden) layers with 20, 60,80, 90 neurons and 1 output neuron for the binary classification.

For our full-feature dataset, the model was trained in 4 epochs with a batch size of 100 records. The network had a 35-neuron input layer (again, the same number as features of the dataset), with similar hidden layers to the model we used to train the 10-best feature version of our dataset (20,60,80,90 neurons) and 1 output neuron for the binary classification.

We initially tested the model with a batch size of 1000, but due to poor performance, we sought a different value. It was observed, that choosing a batch size of 100 records, the specificity of the model was improved.

In both cases, for the input and hidden layers, the activation function that was used was "tanh", while the output layer activation function was "sigmoid". Both tanh and sigmoid activation functions are often used for building Neural Networks, with sigmoid being an ideal choice for binary classification, as its output is within the [0,1] set. Bellow the structure of the LSTM model can be viewed in Figure 3.6.

Table 3.12: Confusion matrices of LSTM models. (10-best feature model on the left, full-feature model on the right).

True\Predict	Normal (0)	Attack (1)	True\Predict	Normal (0)	Attack (1)
Normal (0)	149	328	Normal (0)	430	47
Attack (1)	9139	3658906	Attack (1)	71221	3596824

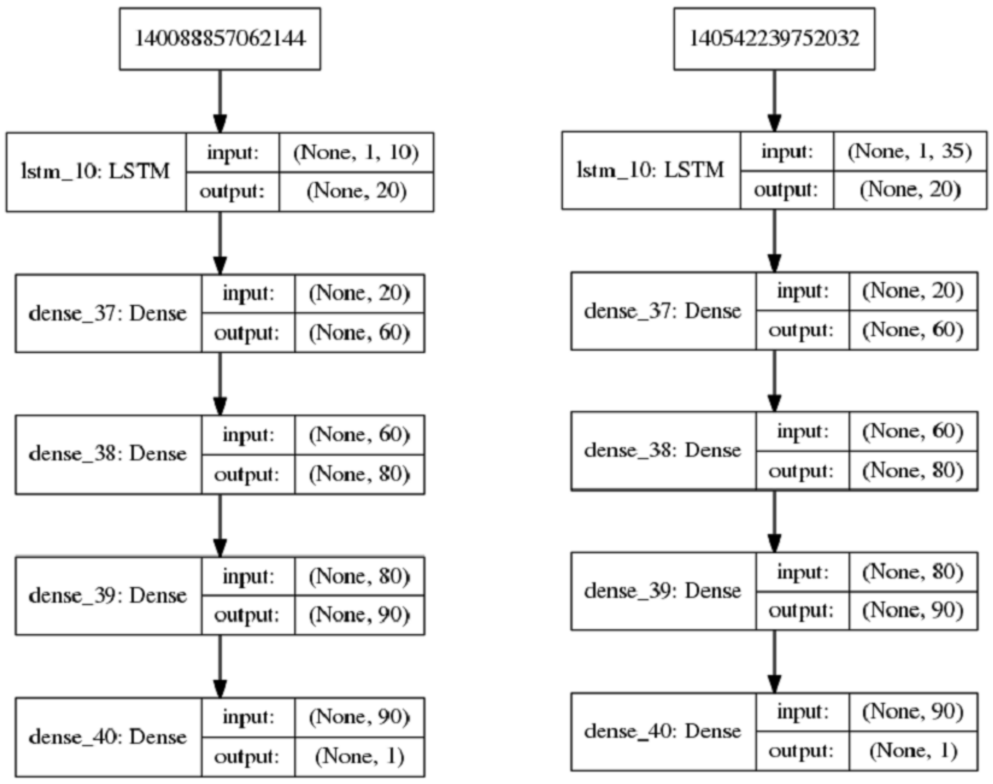


Fig. 3.6: Structure of LSTM model (in layers). (10-best feature model on the left, full-feature model on the right)

Next, we present the training times for both LSTM models, the confusion matrices followed by four metrics.

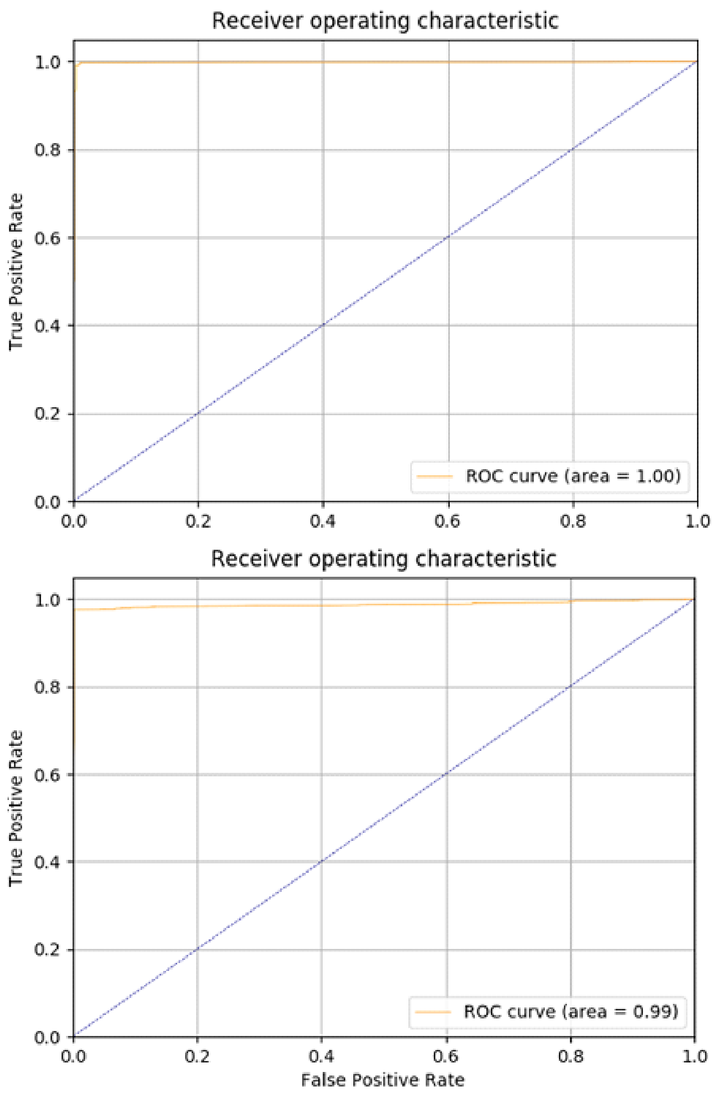


Fig. 3.7: ROC curve for LSTM models. (10-best feature model top, full-feature model bottom)

A.3 RNN

The RNN models were defined to have 1 input layer with the number of neurons equal to the number of input features, two hidden layers and an output layer. For the 10-best features dataset, the Model was trained in 4 epochs (batch size 100), had 10 input neurons (in the first layer, the same number as the features), with hidden layers similar to the ones in the LSTM models, and 1 output neuron for the binary classification. As with LSTM-RNN, the parameters were chosen through experimentation. Higher values of batch size affected the model's specificity, as such, we experimented with lower values.

Table 3.13: Confusion matrices of RNN models. (10-best feature model on the left, full-feature model on the right).

True\Predict	Normal (0)	Attack (1)	True\Predict	Normal (0)	Attack (1)
Normal (0)	127	350	Normal (0)	379	98
Attack (1)	9171	3658874	Attack (1)	76718	3591327

For our full-feature dataset, the model was trained in 4 epochs (batch size 100) and had a 35-neuron input layer, same number and consistency of hidden layers as with the 10-best feature model and 1 output neuron for the binary classification. In both cases, for the input and hidden layers, the activation function that was used was "tanh", while the output layer activation function was "sigmoid". As mentioned previously, the sigmoid function is ideal for output layers for a binary classification problem, as its output is within the [0,1] set. Bellow the structure of the RNN model can be viewed in Figure 3.8.

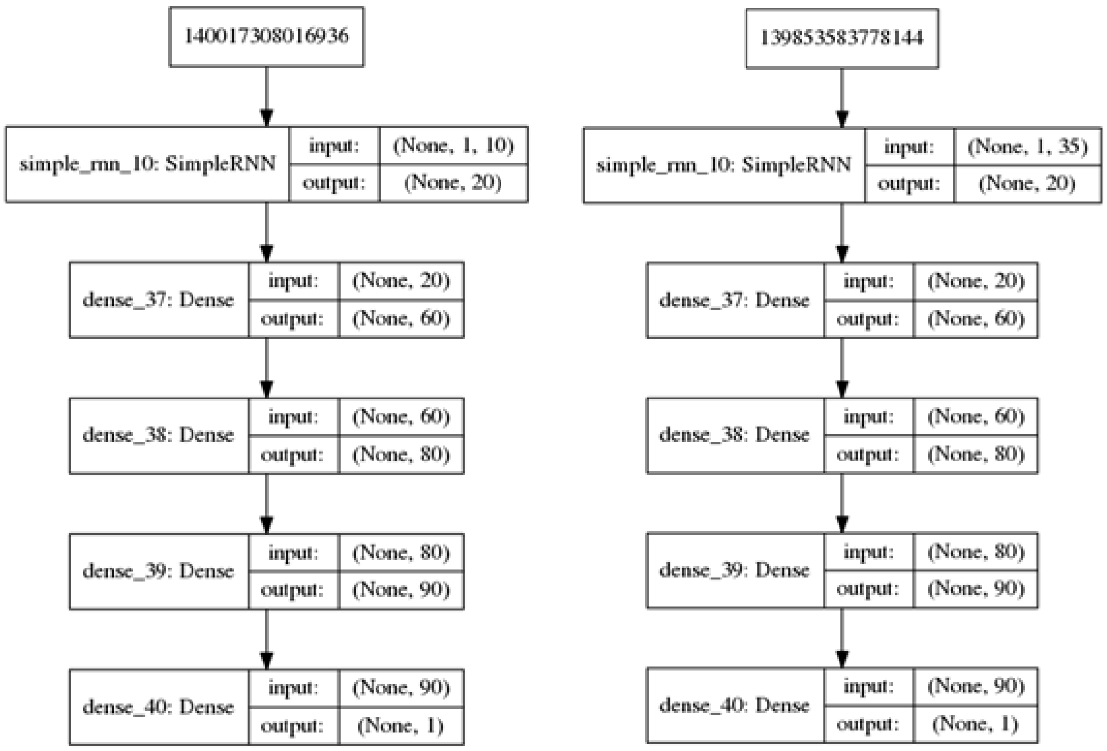
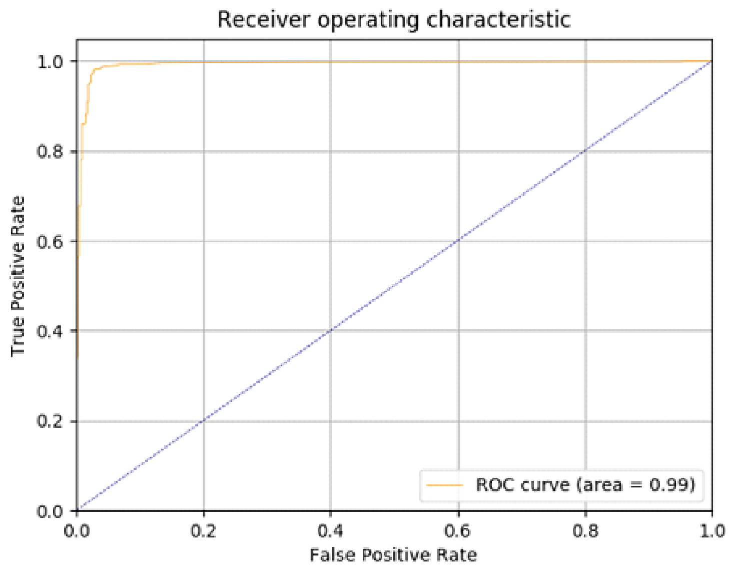


Fig. 3.8: Structure of RNN model (in layers). (10-best feature model on the left, full-feature model on the right)

Next, we present the training times for both RNN models, the confusion matrices followed by four metrics.



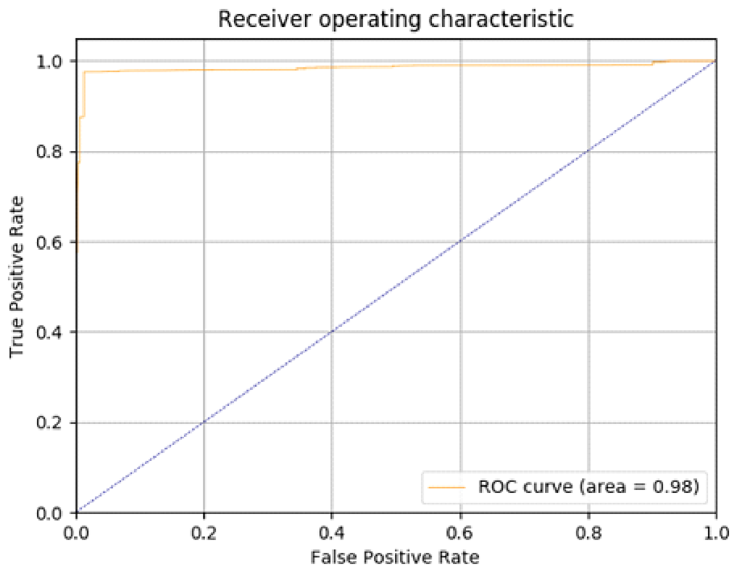


Fig. 3.9: ROC curve for RNN models. (10-best feature model on top, full-feature model on bottom)

Finally, we evaluated a simple Deep RNN network's capabilities of distinguishing between normal traffic and each of the attacks that are described in the dataset separately. The network had 10 input neurons, and 8 hidden layers comprised of 30, 40, 40, 60, 80, 80, 100 neurons each. Finally, the output layer had 1 neuron for binary classification. The resulting confusion matrices can be viewed in Table 3.15. In the following table (3.14), we present the four metrics Accuracy, Precision, Recall, Fall-out along with Training Time of all the models that were trained.

Table 3.14: Accuracy, Precision, Recall Fall-out and training time Metrics. (In tables(b) and (c), the model was an RNN. Training time is in seconds)

(a)	SVM		RNN		LSTM	
	10-best	Full features	10-best	Full features	10-best	Full features
Accuracy	0.88372702	0.99988742	0.99740468	0.97906078	0.9974194	0.9805731
Precision	1	0.99988742	0.99990435	0.99997271	0.99991036	0.99998693
Recall	0.8837119	1	0.99749976	0.97908477	0.99750848	0.98058339
Fall-out	0	0.86582809	0.73375262	0.20545073	0.68763103	0.09853249
Training Time	1270.48	6636.98	8035	6888.08	10482.19	14073.63
(b)	Normal- DDoS HTTP	Normal- DDoS TCP		Normal DDoS UDP	Normal- DoS HTTP	Normal- DoS TCP
Accuracy	0.99317872	0.99991921		0.99999473	0.98063201	0.99993672
Precision	0.99295065	0.99999488		0.99999684	0.98984428	0.99994479
Recall	0.99696663	0.99992429		0.99999789	0.98451178	0.99999188
Fall-out	0.01467505	0.01048218		0.00628931	0.03144654	0.07127883
Training Time	36.05	2250.9		2053.89	38.41	1362.27
(c)	Normal- DoS UDP	Normal- OS Fingerprinting		Normal Service Scan	Normal- Data exfiltration	Normal- keylogging
Accuracy	0.99998645	0.99173509		0.99568199	0.98757764	0.98727273
Precision	0.99999419	0.99842856		0.99859012	0	0.98529412
Recall	0.99999226	0.99307804		0.99706156	0	0.91780822
Fall-out	0.01257862	0.05870021		0.21593291	0	0.00209644
Training Time	2269.61	74.21		188.27	38.41	38.74

Table 3.15: Confusion matrices of RNN models for subcategory evaluation.

True\Predict	Normal (0)	DDoS TCP (1)
Normal (0)	472	5
DDoS TCP (1)	74	977306

True\Predict	Normal (0)	DDoS HTTP (1)
Normal (0)	470	7
DDoS HTTP (1)	3	986

True\Predict	Normal (0)	DDoS UDP (1)
Normal (0)	474	3
DDoS UDP (1)	2	948253

True\Predict	Normal (0)	DoS TCP (1)
Normal (0)	443	34
DoS TCP (1)	5	615795

True\Predict	Normal (0)	DoS HTTP (1)
Normal (0)	462	15
DoS HTTP (1)	23	1462

True\Predict	Normal (0)	DoS UDP (1)
Normal (0)	471	6
DoS UDP (1)	8	1032967

True\Predict	Normal (0)	OS Fingr/t (1)
Normal (0)	449	28
OS Fingr/t (1)	124	17790

True\Predict	Normal (0)	Service Scan (1)
Normal (0)	374	103
Service Scan (1)	215	72953

True\Predict	Normal (0)	Data exfiltration (1)
Normal (0)	477	0
Data exfilt. (1)	6	0

True\Predict	Normal (0)	Keylogging (1)
Normal (0)	476	1
Keylogging (1)	6	67

3.8.4. Overview of three classifiers and discussion of results

Overall, results indicate high accuracy both in Binary and Multiclass classification.

Data exfiltration had the worst metrics of all (in multiclass classification). Training time is somewhat proportional to the records used to train the model, more records, more time for training, the one that took longest to train was DoS UDP-Normal traffic.

Table 3.16: Summarization of models' parameters

		Max Iterations	Epochs	Layers	Neurons	Activation function	Batch size
SVM	10-Best Features	3000	-	-	-	-	-
	Full Features	1000	-	-	-	-	-
RNN	10-Best Features	-	4	6	10 Input hidden layers 20 1st 60 2nd 80 3rd 90 4th 1 Output	Hidden layers: 'tanh' Output layer: 'sigmoid'	100
	Full Features	-	4	6	35 Input hidden layers 20 1st 60 2nd 80 3rd 90 4th 1 Output	Hidden layers: 'tanh' Output layer: 'sigmoid'	100
	Attacks Binary Classifications	-	4	9	10 Input hidden layers 30 1st 40 2nd 40 3rd 60 4th 80 5th 80 6th 90 7th 1 Output	Hidden layers: 'relu' Output layer: 'sigmoid'	~100

LSTM	10-Best Features	-	4	6	10 Input hidden layers 20 1st 60 2nd 80 3rd 90 4th 1 Output	Hidden layers: 'tanh' Output layer: 'sigmoid'	100
	Full Features	-	4	6	35 Input hidden layers 20 1st 60 2nd 80 3rd 90 4th 1 Output	Hidden layers: 'tanh' Output layer: 'sigmoid'	100

Additionally, in the binary classification, Fall-out values were rather high, with the exception of RNN and LSTM models for the fully-featured version of the dataset. This could be explained by a number of factors, such as poor optimization of models in use and the relatively low number of epochs that we chose, in order to speed up the process. In table 3.16, the parameters that were chosen for the three models are presented. The obtained results indicate that the proposed dataset can be used to train accurate classifiers, with Recurrent Neural Networks and their permutation (LSTM) outperforming the SVM implementation. Additionally, models trained on the chosen 10-best feature version of the dataset displayed higher accuracy than the full version.

3.9. Conclusion

This chapter presents a new dataset, Bot-IoT, which incorporates both normal IoT-related and other network traffic, along with various types of attack traffic commonly used by botnets. This dataset was developed on a realistic testbed, and has been labelled, with the label features indicated an attack flow, the attacks category and subcategory for possible multiclass classification purposes. Additional features were generated to enhance the predictive capabilities of classifiers trained on this model. Through statistical analysis, a subset of the original dataset was produced, comprised of the 10-best features. Finally, four metrics were used in order to compare the validity of the dataset, specifically Accuracy, Precision, Recall, Fall-out. We observed the highest accuracy and recall from the SVM model that was trained on the full-featured dataset, while the highest precision and lowest fall-out from the SVM model of the 10-best feature dataset version. With further optimization of these models, we argue that better results could be achieved. In Chapter 5, we present the development of a network forensic model using deep learning and evaluate its reliability using the BoT-IoT dataset. In the next chapter, we provide the machine learning evaluation of the Bot-IoT dataset.

Chapter 4

Role of Network Forensic techniques using machine learning in IoT environments

4.1. Overview

Since technological advances allow for the miniaturization of hardware, the Internet of Things (IoT) applications have significantly increased over the years. With reports indicating that by 2020, the number of active IoT devices around the globe will reach 20.4 billion [7], showing an increase of 145% from 2017, it is evident that the impact of IoT devices in the market will only increase, as industries discover the benefits of automation that IoT services can offer. However, as the IoT's popularity increases, its security is brought into question. A 2015 study by Hewlett Packard [10] concluded that several IoT devices that were investigated were deemed insecure, with 80% raising privacy concerns, while 60% lacked mechanisms for verifying and authenticating security updates. These weaknesses can be exploited, enabling an attacker to stealthily modify a device's firmware. Furthermore, real-world examples of attacking the IoT have already been observed. Possibly the most famous such example was the Mirai botnet. Mirai belongs in a family of new botnets that primarily target smart things (IoT devices) due to their weak security measures and always on-line design [68]. The first time the Mirai Botnet was observed, it was launching DDoS attacks that were targeting Brian Kreb's blog, with the first wave of DDoS attack peaking at 623 Gbps (77.9 GBps) and latter escalating against the French web-host and cloud service provider OVH reaching 1.1 Tbps [68].[1]

It is widely accepted that the Internet is filled with security risks, where consequences may range from a simple inconvenience to serious cyber threats. One famous security threat that utilizes the Internet in its architecture, and is considered to be quite destructive is the botnet [211]. Botnets are collections of infected

[1]Part of the work presented in this chapter has been published in:
Koroniotis, N., Moustafa, N., Sitnikova, E. and Slay, J., 2017, December. Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques. In International Conference on Mobile Networks and Management (pp. 30-44). Springer, Cham.

devices that are organized by a botmaster and can be used to launch various diverse and destructive cyber attacks. Recently, a number of well-known incidences, where botnets used the aggregated processing capabilities of the IoT appeared, revealing a number of new infection routs. For instance, attackers task-specific bots with actively scanning the Internet to discover vulnerable devices, gain access by breaching their security and then allow other specialized bots to infect them with the botnet's malware [136]. IoT-specific rout of exploitation, attackers take advantage of close proximity networks, thus gaining access to and direct communication to an IoT network, infecting a vulnerable device, and enabling the propagation of the malware to other IoT devices in the immediate vicinity [26]. Thus, investigating network-based attacks that target IoT systems is essential.

The Botnet scenarios mainly include three stages: the botmaster, the Command and Control (C&C) infrastructure and the bots (infected machines), to launch sophisticated advanced persistent threats [212, 213]. A botmaster commands a network of bots, receiving reports and transmitting commands through the C&C infrastructure. As such, botnet investigations may focus on either identifying the communication channels used by the C&C, or the attacks launched by the botnet. Detecting attacks launched by a botnet is often difficult, as botnets are often designed to launch a number of diverse malicious events, such as DDoS, Keylogging, Phishing and Spamming, Identity theft and even proliferate the malware of other botnets [22], with some attacks appearing to be similar to normal user traffic (some DDoS for example). Furthermore, botnet attacks can be very destructive, causing entire sections of the Internet to collapse for hours (Mirai botnet) or even steal credentials for online accounts and banking (Zeus botnet) [214]. Thus, multiple diverse methods are used to investigate botnets, such as Network Intrusion Detection Systems (NIDS) which could employ anomaly-based detection. However, those systems can not trace the origins of botnets and demand a large number of legitimate observations to build a reliable detection engine.

Network forensics systems would be the appropriate solution to discover and trace botnet events from IoT networks [215]. Network forensics is a sub-discipline of the digital forensics which specializes in monitoring and inspecting network-related sources of data (e.g., logs, network data flows, payloads) for detecting the origins of security incidents and policy violations [38]. During investigations, multiple networks and IoT devices need to be scanned, with considerable volumes of captured traffic often processed in order to identify traces of an attack. This process is quite difficult and time-consuming for humans to do by hand, which leads forensic experts to employ automated solutions such as machine learning models. Multiple studies have produced network forensic solutions based on machine learning models, harnessing their automated processes to rapidly process data to detect botnet activities even in real-time [213].

Most of existing network forensics tools use expert systems instead of machine learning, with prime examples being NIDS [216]. Expert systems use if-then-else rules to represent knowledge, which is rather limiting and is greatly reliant on human experts to encode their knowledge in rule form for the system to use [217]. Machine Learning (ML), on the other hand, analyses behaviour dynamically, identifies patterns in the data through statistical methods. which does not require the actions of an expert to train and evaluate its performance. ML only demands to adopt feature engineering, and training and validating appropriate classifiers to data collections and can be re-trained easily on new data.

In this chapter, we focus on evaluating the performances of supervised and unsupervised machine/deep learning models as network forensics models. The aim of this evaluation is to identify and trace botnet events from IoT networks. We utilize five supervised ML algorithms: decision tree, artificial neural network, naive Bayes, association rule mining, deep belief network, and two unsupervised ML algorithms: k-means and expectation-maximization, for examining their performances as the first stage of network forensics. The network forensics techniques are assessed on three benchmark network datasets, NSL-KDD, UNSW-NB15, and Bot-IoT.

The main points of this chapter are as follows:

- Analysis of big data collections, such as NSL-KDD, UNSW-NB15 and Bot-IoT, for investigating attack events and their traces for developing reliable network forensics models in IoT networks.
- The application of several machine learning/deep learning techniques using network data sources as network forensics models, in order to determine their credibility of developing network forensics.
- Assessment and comparison of network forensics, using machine learning for discovering botnet activities from IoT networks.

Chapter 4 is structured as follows. The literature review of the study is discussed in section 2. In section 3, network forensic architecture of the process depicted in this chapter is presented. Sections 4 depicts the three datasets used in this chapter, the feature selection process, a brief description of the machine learning algorithms followed by a discussion regarding their performance in section 5. Concluding remarks are given in section 6.

4.2. Background

4.2.1. Internet of Things

Although the IoT is selected as the preferred technology by organizations around the world, seeking to take advantage of its remote management features and increase productivity while lowering costs of operation, it is unmistakable that several issues related to security and privacy persist. For instance, Ronen et al. [26] sought to investigate possible attack vectors for the Philips Hue smart lamp. These smart lamps primarily utilize ZigBee as the local communication protocol which enables the communication between lamps and their controller. The researchers implemented an attack vector for a particular brand of a smart lamp, which took advantage of security weaknesses present in the devices, allowing for their exploitation. Furthermore, they proposed that if a large enough number of smart lamps were distributed around a city and their relative distances did not hinder their communication, a malware could propagate throughout the city, “jumping” between devices and allowing for further infection of otherwise secure networks.

Similarly but choosing a broader point of view, Hossain et al. [1], provided an analysis of existing security issues in the IoT. By reviewing the literature, the researchers were able to assert that smart things were vulnerable, as many examples existed where researchers were able to compromise a device’s security with ease. Next, and by considering the IoT ecosystem which consists of IoT devices, Sensor Bridge Device, Coordinator, IoT Service and Controller, they indicated the inapplicability of conventional cybersecurity principles and forensic mechanics. The reasons behind the need for new forensic and security mechanisms were the inherent constraints of smart devices such as limited battery life, processing power, mobility and the diversity of technologies used for similar IoT implementations. Thus, the researchers concluded that, as there exists a gap in these areas, both cybersecurity and forensics for IoT should become a priority.

On a different note, Pa et al. [136], investigated the types of IoT devices that are receiving active scans from the Internet, using their observations in order to construct IoTPOT, an IoT-specific honeypot. The proposed honeypot incorporated a module that grouped several virtualized IoT devices which were called IoTBOX. The main purpose behind IoTBOX was to render the proposed IoTPOT and thus the virtual devices it utilized as realistic as possible. By deploying the IoTPOT, the researchers were able to observe several strains of malware in action. One noteworthy repeating pattern was that in some malware infections, initially, a host would perform information gathering and the intrusion, after which the detected credentials would be forwarded to a second host, which took care of the actual infection. Similar to Hossain et al. [1], a literature review revealed that the field of IoT, its forensics and its security are under constant development, with new vulnerabilities discovered on a daily basis.

4.2.2. Botnets in IoT Networks

In order to detect and investigate botnets and their activities, researchers have proposed a number of diverse techniques over the years. A study of the Citadel Botnet, which however is not an IoT Botnet, was conducted by Rahimian et al. [218]. In their analysis, the researchers applied several code analysis methods, in order to gauge the similarities between the Zeus and Citadel botnets, as the Citadel was spawned from Zeus. The similarities were then utilized in order to speed up the reverse engineering process, through a process they proposed and named clone-based analysis. Through the clone-based analysis, parts of the malware code that were originally discovered in different malware are identified and as such, the portion of code that needs to be reviewed is reduced. A tool named BotMosaic was introduced by Houmansadr et al. [219], with its main functionality focusing on Botnet traffic detection and type identification. BotMosaic detects IRC Botnets by utilizes a watermarking technique for NetFlow traffic, which is non-distorting thus marking traffic without altering it for later identification. Even though many researchers have developed various techniques for the identification, investigation and fingerprinting of Botnets, malware programmers constantly adapt their code, making their Bots more resilient against identification and takedowns. This and along with the nature of IoT devices, cause issues for security experts, that need to be addressed.

Malware authors equip their Botnets with a number of diverse attacking activities, such as DoS, DDoS, Phishing, Spamming, Identity theft and malware proliferation [22]. Since Botnets have existed for some time, considerable research has been done in the creation of techniques focusing on detecting and identifying the presence of botnet activities in a network. For example, several techniques that have been proposed rely on machine learning for the analysis of captured packets, either in packet form or grouped in network flows, and are utilized for discrimination of normal and botnet network traffic [220]. The delivery of botnet malware to a remote target is carried out through a collection of mechanisms called propagation mechanisms. Typically, propagation techniques can be grouped into two types, active and passive. Active techniques, also known as self-propagation, rely on the botnet to scan the Internet for devices that display vulnerabilities and exploit them, adding the infected host to the botnet [70]. On the other hand, passive techniques rely on user interaction, specifically, a user has to access compromised social media, storage or websites, which causes the bot malware to be downloaded and installed on the victim's machine, thus causing the machine to become a part of the botnet [70].

A number of studies [148, 221, 222] have taken advantage of machine learning techniques for the design of intrusion detection models, which can differentiate between normal traffic and Botnet activities. To that effect, Roux et al. [148] designed an Intrusion Detection System (IDS) comprised of several probes that record wireless transmissions in an IoT network. The proposed IDS collects relevant features such as signal direction and strength, which are used to determine whether a detected signal originated from outside the monitored network, in which case, the signal is determined to be suspicious. The classification of signals into legitimate and attacks is performed by a neural network. A different implementation that relied on a Support Vector Machine (SVM) to detect botnet traffic was developed by Lin et al. [223]. The proposed method made use of Artificial Fish Swarms to determine a set of optimal features for the SVM. The researchers noted a significant improvement time-wise, although the SVM's accuracy was slightly better than when the Genetic Algorithm was used. On a similar note, Greensmith et al. [224], investigated the usage of Artificial Immune Systems (AIS) in an attempt to secure smart things. The researchers proposed that a viable way to handle the heterogeneous nature of IoT was through the use of multiple AIS.

4.2.3. Existing Network Forensic Tools

The development in the area of network forensics has produced various specialized commercial and open-source tools that are being used by forensic professionals and network security practitioners alike [225, 226, 227, 228]. The most common network forensics tools are discussed in the following:

- **NetworkMiner** [225] is a tool designed primarily for the Windows Operating System (OS), but can also function in some versions of Linux/Unix OS. The main function of this tool is as a passive packet sniffer for the detection of, among other things, OS, sessions and open ports. It can also be used as an off-line pcap parser, providing file and certificate extraction services. One key characteristic of NetworkMiner is that its service detection is independent of the port number, which is achieved through statistical models.
- **Xplico** [229] is an analysis tool designed for the Ubuntu OS, similar to NetworkMiner in that it can be used to extract files from pcap files. Its main function is to identify the protocol of network flows through a process similar to NetworkMiner, and extract files like e-mails, conversations from VoIP, web pages and more. Some protocols that Xplico can detect include VoIP, HTTP, IMAP, POP, SMTP, FTP. A very helpful mechanism of this tool is that it associates the re-constructed files with the network flows from which it was extracted.
- **PcapXray** [230] is an offline network traffic analysis tool, that extracts information from pcap files. Primarily a visualization tool for Linux OS, PcapXray highlights Tor and malicious traffic, providing a diagram of the identified connections. In the produced diagram, network devices are represented as nodes, connected with each other through the identified network connections. To identify any potential malicious traffic, PcapXray relies on rule-based mechanics.
- **Zeek Security Monitoring (previously named Bro)** [226] is an open-source traffic analysis tool which originally functioned as an IDS but has had its functionality extended. It is versatile, by using Bro's scripting language, users are able to define custom filters to detect various malicious scenarios, although its default behaviour is to rely on prepared patterns of well-known attacks. Additionally, Bro can be connected to other network security entities, like firewalls through its NetControl framework, allowing it to block attacks when they are detected.
- **Prads** [231] is a passive real-time system used for detection of devices in a network. It is used to map a network by passively collecting information

about its devices through the network traffic it processes. Prads collects information about host machines and services that are 'live', along with some limited traffic statistics. Protocols that are used for both OS and service fingerprinting, include TCP, UDP, ICMP, while ARP can also be used for host detection.

- **Tcpstat** [232] is a command-line network analysis tool primarily tested on Linux machines. It receives as input either a network interface, performing analysis on live traffic, or by parsing pre-recorded pcap files. Furthermore, it can display various statistics for TCP and UDP connections, including bandwidth, number of packets and packets per second. Snort [227] is another open-source IDS that can function in three modes, sniffer mode, packet logging mode and NIDS mode. It utilizes rule-based detection for identifying compromises, like Bro and PcapXray.

Many of the aforementioned tools utilize signature-based detection, relying on predefined knowledge rules to identify attacks in traffic. Signature-based tools often display a high detection rate for known attacks but are incapable of generalization in order to detect new attacks [233]. Furthermore, they and can be fooled by attacks, the patterns of which have been altered by the attackers. Although most available tools base their functionality on knowledge rules, machine learning methods are more versatile [234]. Through machine learning, mathematical models are built on known data through automated means. Thus, instead of relying on an expert to craft rules to describe events that a system will flag as malicious, machine learning models rely on the patterns in the data to determine when an attack is present. Furthermore, machine learning approaches can more easily be updated by re-training a model in new data, while for knowledge-based systems, the position of new rules needs to be carefully considered.

4.2.4. Previous Studies of Network Forensics in IoT

A significant issue faced by Network Forensics is the constrained obtainability of network-related evidence. Initially, as IoT devices often are characterized by constrains to their resources, any recorded data is stored locally for a short period of time, before it is transmitted to the cloud backend [235]. Furthermore, the high speeds achieved by today's Internet, means that network traffic is in constant motion which may cause problems during collection [236]. A number of studies have tackled the various challenges of network forensics in the IoT.

To begin with, Khattak et al. [237] handled the storage space issue that arises when working with ever-expanding network log files, by using the implementation of MapReduce in Hadoop. Focusing on network traffic analysis, Bansal et al. [124] proposed a generic Network forensics framework for Botnet detection. Although the proposed framework incorporates established techniques for packet capture and inspection, it remains on a theoretical level. On the other hand, Saied et al. [213] produced a detection method that relied on inspectors distributed among different networks that focused on identifying DDoS attacks. The proposed method relied on an Artificial Neural Network that processed timing and packet header data, collected from the distributed inspectors. Another system that focused on DDoS detection was proposed by Divakaran et al. [113]. Initially, the packets were organized into a session, with each session containing multiple network flows, grouped by packet arrival time. Then, by using a regression model, irregular patterns in the recorded network traffic were identified, indicating a DDoS attack, with the researchers providing additional information on real-world Botnets that were detected during their experiments.

Focusing on malware and the threats that they pose, Wang et al. [174] developed detection and forensic system comprised of multiple cooperating modules. In the proposed system, initially, an attack is detected by a specialized incident detection module, which then starts gathering information by invoking a honeypot. The collected information is then forwarded to a forensic module, which initiates a pre-determined forensic procedure. By utilizing a forensic model after an attack has been detected, the forensic process takes a more active role than usual. Gandhi et al. [140] developed an IoT honeypot named *HIoTPOT*. In this system, a raspberry pi would identify login attempts with unknown credentials to a virtualized IoT device. In such an event, the login attempt would be recorded and an alert issued to the owner of the device. Meidan et al. [144] worked on discovering IoT devices that are part of a botnet. Their method involved the creation of an autoencoder for each of the IoT devices that they used. These autoencoders were trained on the normal behaviour of the devices, flagging any anomalies as bot activity.

Doshi et al. [33] investigated methods for discovering IoT devices engaging in DDoS attacks. This research identified features and traffic statistics that distinguish IoT from non-IoT generated traffic. It was shown by training machine learning models that real-time identification of IoT devices in a DDoS attack can be achieved. Luo et al. [141] proposed an automated method for crafting intelligent-interaction honeypots. The process involved a module probing legitimate IoT devices with attack requests that had been previously collected. The acquired responses were then used in constructing a realistic profile, which was generated by training a machine learning model.

Shone et al. [159] developed an IDS based on a stacked non-symmetric deep autoencoder and a random forest. The encoder segments of two trained autoencoders were stacked, with the second one connected to the random forest classifier. Results indicated an accuracy of 89%. Van et al. [238] incorporated a DBN in a NIDS. In order to identify the best architecture, both a Stacked Autoencoder (SA) and a Stacked Restricted Boltzman Machine (SRBM) were trained, and their classification performances were compared. The SA outperformed SRBM, though it required more training time. Brun et al. [239] investigated attacks against IoT gateways. Specifically, they trained a random neural network to detect Denial of Service and Sleep attacks. The accuracy of the trained model was characterised as similar to that of a threshold detector.

The work we present in this chapter differs from what has already been produced and depicted in this section. Initially, most of the research focuses on either DoS or DDoS attack detection, whereas in our work, various attack scenarios were represented in the datasets, including DoS and DDoS. Furthermore, we evaluated both supervised and unsupervised ML models, to identify the best family of models to use for network forensics, whereas other research focused on supervised classification. We employ a novel optimization method that incorporates Particle Swarm Optimisation to identify an optimal set of hyperparameters for our deep learning model. Finally, it should be noted that our work is intended to be used as a network forensic framework tool, used for the analysis of network traffic by both forensic and security experts alike.

4.3. Proposed Network Forensic Methodology

The proposed network forensics methodology includes four components: network traffic collection, network feature selection, machine learning techniques, and evaluation metrics, as depicted in Figure 4.1 and explained below. The proposed methodology differs from conventional NIDS as contrary to a NIDS' function which is tasked with detecting attacks, it not only discovers attacks but also identifies their origins by using the source and destination IP addresses, which are present in the scanned traffic.

Some attackers employ techniques that mask the identity of their machines, to avoid being identified. One such technique is IP spoofing. IP spoofing is primarily an obfuscation technique, where an attacker forges parts of the header of IP datagrams [240]. Specifically, the source IP address is altered to either a dummy IP or another legitimate address. Apart from hiding their identity, IP spoofing enables some network attacks, like reflection or amplification DDoS attacks [241].

The proposed methodology utilizes network flow-based analysis, where network traffic is grouped into flows of communication, each flow representing communication between two hosts, using a known protocol at a given time. As this approach relies on flow features and statistics, instead of the packets' payload, it is not affected by whether the payload is encrypted or not. As such, our approach is reliable in identifying network attacks. Furthermore, network flow metadata information could be used to identify attack origin [242].

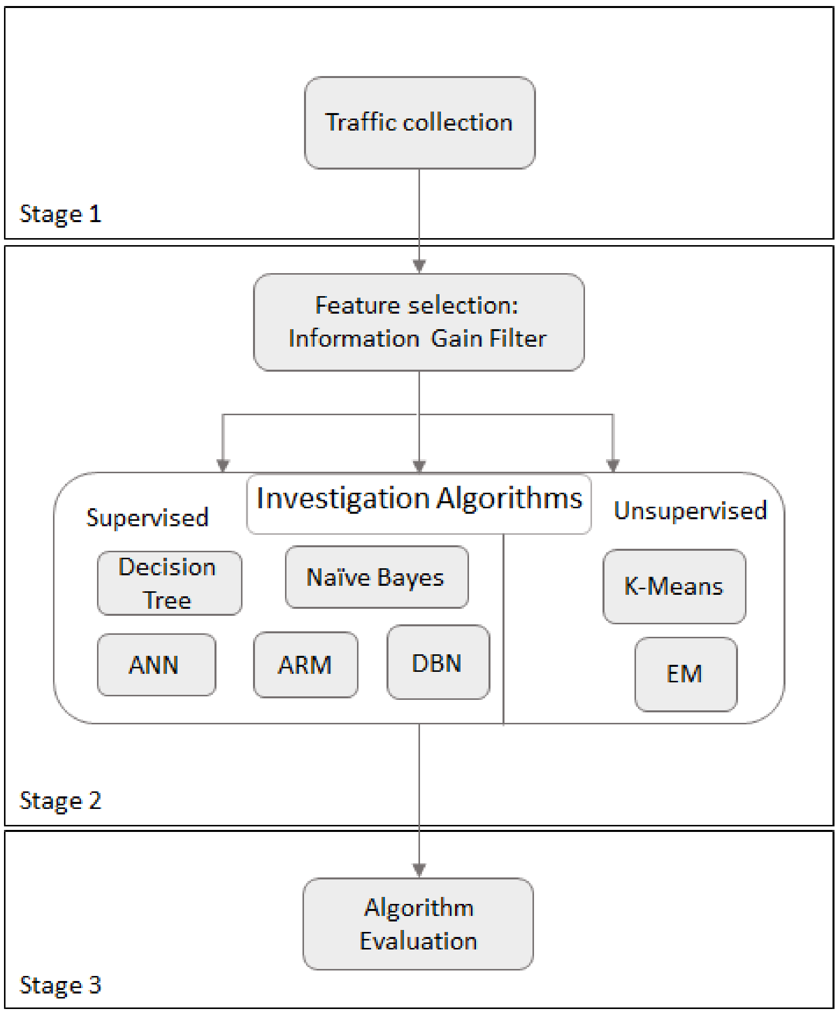


Fig. 4.1: Proposed Network Forensic Methodology

4.3.1. Network Traffic collection

Due to contemporary networks having a tendency to generate considerable volumes of traffic, it becomes a necessity to identify efficient ways to aggregate the packets that are captured, thus improving storage capacity and subsequent use in the construction of network forensic mechanisms. In order to capture and store network packets, the network interface card is accessed by the tcpdump tool [57]. Next, feature generation is performed by forwarding the collected raw network traffic to the Bro and Argus tools. This methodology was employed in order to generate the features of the UNSW-NB15 dataset [57]. For the Bot-IoT dataset, a similar process was followed, where the T-shark tool [65] collected raw pcap files which were then processed by Argus, loaded in a MySQL database and combined into a single relational table, where new features were also generated.

The network collection process needs to be conducted at strategic locations of a network, such as ingress routers so that relevant network flows can be recorded. The key features that identify a network flow include source and destination IP addresses and ports, the type of protocol used and temporal data. By carefully selecting the location where the traffic collection process will be carried out, the investigation of cyber incidents is sped up. The traffic collection process for the Bot-IoT is given in Figure 4.2.

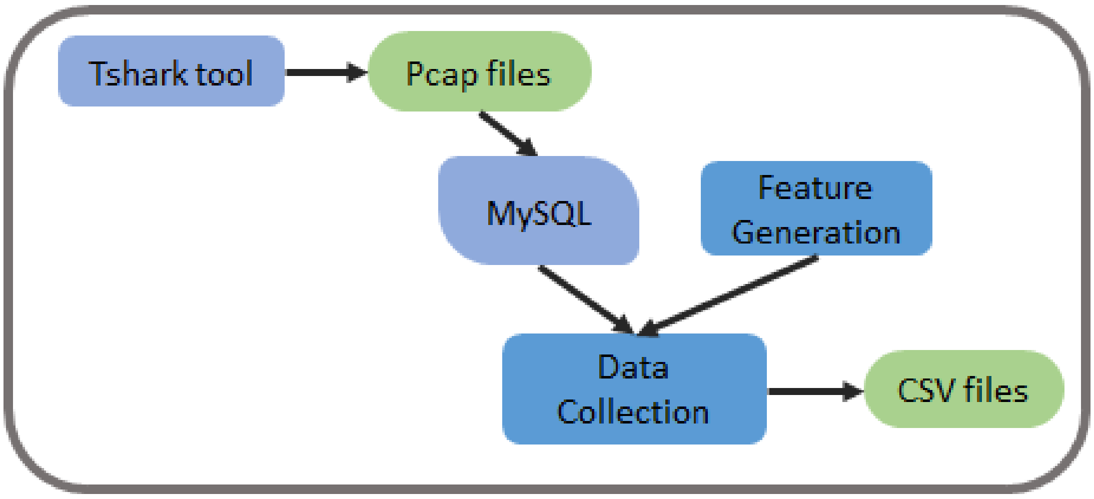


Fig. 4.2: Bot-IoT Network Feature Collection

4.3.2. Network Feature Selection

Datasets are usually comprised of numerous features, each providing some information about the data that is represented. However, not all features contribute equally to the classification process. As such, methods such as feature selection are employed, in order to identify important attributes to improve the classification accuracy. Feature selection techniques are separated into filter, wrappers and combinations of the two named hybrids. Filter methods utilize statistical techniques to measure the relation between features and the class feature while, on the other hand, wrapper methods rely on underline Machine Learning technique [243].

Multiple filtering methods have been proposed in the literature, with two examples being Information Gain (IG) and Chi-Square x^2. In contrast, wrapper methods focus on the performance of the Machine Learning model that is to be trained on the data in question, as a way of determining the optimal feature subset. Specifically, wrapper methods create sub-groups of the available features and train the intended model, recording the error rate and thus measuring the effectiveness of the features sets. A combination of the wrapper and filter methods results in hybrid methods

In this chapter, the IG filter method is utilized due to its simplicity and ability to identify useful features in large volumes of data, as is often the case with network datasets. A key concept in IG, and thus in Information theory, is entropy. Entropy is a mathematical expression of the “unpredictability” or dissimilarity of the probability distribution of a collection of observations [244]. Simply stated, in the context of machine learning and feature selection, entropy measures the amount of “information” that a feature displays about the class function.

Higher values of entropy equate to larger dissimilarity in the sample space. The entropy H(Z) of a feature “Z” is given in Equation 4.3.1, where P(z) is the probability of class “z” in the group “Z”. It should be noted that when p(z) is equal to 1, which means there is no uncertainty, H(Z) is 0, while when all classes are likely $(p(z_1) = p(z_2) = p(z_n))$ then H(Z) is maximized, thus the values of H(Z)>=0. The second important calculation necessary to define IG is conditional entropy [245]. Conditional entropy is defined as the uncertainty of a feature “K” given another feature “Z”. Equation 4.3.2 gives the conditional entropy of “K” given feature “Z” takes value “z”. To get the conditional entropy of the two features for all their values, the entropy equation becomes as shown in Equation 4.3.3.

$$H(Z) = -\sum_{z}(p(z) * \log p(z)), z \in Z \tag{4.3.1}$$

$$H(K/Z = z) = -\sum_{k}(p(K = k/Z = z) * \log(p(K = k/Z = z))), z \in Z, k \in K \tag{4.3.2}$$

$$H(K/Z) = \sum_{z}(p(z) * H(K/Z = z)) = -\sum_{k,z}(p(k,z) * \log(p(k,z)/p(z))), z \in Z \tag{4.3.3}$$

$$IG(Z) = H(K) - H(K/Z) \tag{4.3.4}$$

From the previous two entropy calculations, Information Gain is produced. Information Gain for a feature “Z” is defined as the reduction of entropy of a dataset when feature “Z” is used to split the data [244]. The value of IG is calculated for individual features, is produced by using equations 4.3.1 and 4.3.3 and is given in Equation 4.3.4 where IG(Z) is the Information Gain for feature “Z”, H(K) is the initial entropy of the data and H(K/Z) is the conditional entropy of the data for feature “Z”. One important application of IG is for the construction of decision trees [246]. Specifically, a decision tree uses IG to select the best feature to use in the splitting/branching process. In the case of feature selection through ranking, the IG for each feature is calculated, in order to measure how well the class feature is separated. Higher values of IG indicate a greater reduction in entropy and thus a contribution to classification for the feature in question, while values close to or equal to zero denote a minimal to no change in entropy. We used IG on all three datasets, maintaining the 10 best features, in other words, the features with the highest IG rank.

4.4. Machine learning techniques

In order to investigate five well-known supervised and two unsupervised Machine Learning (ML) algorithms, the Weka [247] and Python programming are used. The supervised ML algorithms learn patterns by processing data for which the class feature, in other words, the target of the classification, has been provided. The five supervised ML algorithms used in this work are discussed as follows:

4.4.1. Association Rule Mining (ARM)

ARM is defined as a machine learning technique that is used for the identification of associations between features in data. The rules discovered by ARM are called association rule and are in the form of *if-then*. They can be interpreted as "if A is present, then B is present", where A and B are values in the data [248, 249]. Association rules are defined as having two "itemsets", the left-hand-side (LHS) and the right-hand-side (RHS) set, with an example given in Equation 4.4.1.

$$X \Rightarrow Y, X, Y \subseteq D \tag{4.4.1}$$

Where "X" is the LHS, "Y" the RHS of the rule and "D" represents the data. Two very important metrics that are used by ARM algorithms in the rule creation process are *support* and *confidence*. Support indicates the fraction of instances in D where an itemset X appears, where X ⊆ d and "d" being a full record of D, given in Equation 4.4.2. While confidence is used to indicate the occurrence of the rule that is being investigated, compared to the occurrence of the LHS, given in Equation 4.4.3.

$$support(X) = |d, X \subseteq d|/|D|, d \subseteq D \tag{4.4.2}$$

$$confidence(X \Rightarrow Y) = support(X \cap Y)/support(X), X, Y \subseteq D \tag{4.4.3}$$

During the training phase of an ARM, frequent itemsets with a support value that exceeds a user-defined minimum threshold are sought. These itemsets are used to form rules, the confidence of which needs to exceed a user-defined minimum threshold. The process continues until all frequent itemsets and association rules are found. Various algorithms have been proposed over the years, with the most famous example being *Apriori*.

The Apriori algorithm works by iteratively finding frequent itemsets, increasing their size by one item for each iteration, until a point is reached where no further frequent itemsets can be found. It should be noted that Apriori works under the assumption that infrequent itemsets can not be combined to have frequent sets, as such they are removed. In this chapter, we used OneR [250], a very simple ARM algorithm that makes predictions using a single feature. The algorithm functions by producing a single rule for each feature in the dataset, based on the frequency of appearance, and keeping the rule which displayed the smallest error.

4.4.2. Artificial Neural Networks (ANN)

Artificial Neural Networks are a category of ML models, inspired by the processes of the human brain [209]. They are part of the family of parametric classifiers, having parameters such as: number of neurons/layers, weights and biases, and a collection of hyperparameters like: number of epochs, learning rate and batch size. A typical Neural Network (NN), can be depicted as a directed, usually acyclic graph, whose nodes are called *neurons* and edges *synapsis*, with neurons organized into layers. Typically, a NN is comprised of an input layer, some hidden layers and an output layer, with an example given in Figure 4.3.

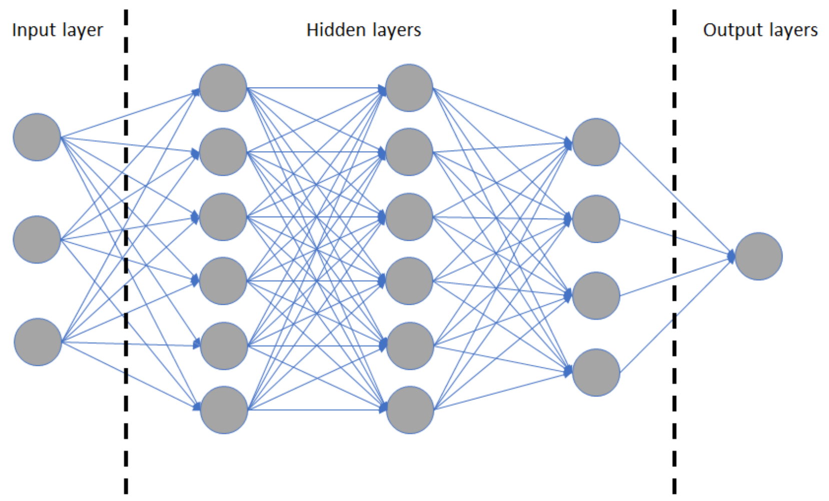


Fig. 4.3: Artificial Neural Network example

A neuron of a layer has multiple inputs from previous layers, which are summed up and passed through an activation function. The resulting value determines whether the neuron "fired" or not. Training is usually performed through a process known as *backpropagation* [251]. During backpropagation, after the NN has estimated the class feature's value for a given input, a *cost function* is utilized in order to estimate the error of the NN. After the error has been estimated, the weights and biases (starting from the output layer backwards) are then corrected (updated), by adding or subtracting the partial derivative of the cost function, with respect to weights and biases, times the learning rate. Over the years, a number of different types of NNs have emerged, which slightly alter the networks processes or structure, improving performance when applied to certain problems.

- Multi-Layer Perception [252]: A Multi-Layer Perceptron is an example of a simple *feed forward neural network*, distinguished from a linear perceptron by the choice of activation function and number of layers. It is comprised of an input layer, a collection of hidden layers and an output layer. Its neurons employ non-linear activation functions, and it is trained through backpropagation.
- Recurrent Neural Networks [210]: These types of Neural Networks incorporate a form of memory in their structure. In their simplest form, an RNNs output is influenced by not only its input during its current iteration but also the hidden state of the previous iteration. As such, RNNs, in contrast to other NNs, are suited to processing data that may exhibit temporal relations, such as the data in the Bot-IoT dataset. Some popular applications for RNNs include automatic image generation, object identification, speech recognition and automatic translation. An improved version of RNN has been in used in recent years, named LSTM (Long-Short Term Memory), where inside the RNN Neuron, an extra mechanism is employed, that maintains a sporadically updated memory of previous outputs[210].
- Voted Perceptron [253]: This version of ANN is a variation of the vanilla perceptron algorithm, where classification is performed through voting of all the trained perceptrons. During training, the number of data points that elapsed before a perception's weight was updated, is maintained and used as a weight. The produced weights function as a number of votes for each perceptron, with all existing perceptrons used during classification time [254]. In this chapter, we employed Voted Perceptron, for its reported performance over the vanilla perceptron [253].

4.4.3. Naïve Bayes (NB)

Naïve Bayes is a family of supervised statistical classifiers, which make use of probability theory. These models rely on the calculation of the *a posterior probability* of each class, given a sample, in order to perform classification [255]. During training, estimates of these probabilities are calculated from the training data, which are then used during the classification process. This probability is given by applying the Bayes' Theorem which is given in Equation 4.4.4.

$$P(Y/X) = P(X/Y) * P(Y)/P(X) \tag{4.4.4}$$

The above equation can be interpreted as “given a sample X, the probability of that sample’s class value being Y, is equal to the conditional probability of X given Y times the probability of class Y occurring, divided by the occurrence of sample X” . Classification is performed for a record R_1 comprised of some features, into a specific class C_2, if the conditional probability of record R_1 to belong to class C_2 exceeds the conditional probability of that record to belong to another class. This can be expressed with the following inequality of probabilities, $P(C_2/R_1) > P(C_n/R_1)$, with C_n being a class other than C_2.

The term “naïve” indicated the assumption, unrealistic though it might be, that the features are independent of each other and each contributes to the class prediction separately [251]. Although this assumption may be erroneous, in practice it has been shown that Naïve Bayes achieves relatively good results for large enough datasets. After calculating the posterior probabilities for each class, the model assigns the class with the highest probability to the new sample.

Although Naïve Bayes classifiers are generally simpler to built and train than others, they can process data fast, which makes them applicable to large datasets, and their results can be easily interpreted, they too have some disadvantages. In practice, the assumption of independence between features may hinder the classification process if not enough data is given, leading Naïve Bayes to display less accuracy compared to other classifiers [251]. Furthermore, their performance tends to be worse than that of other classifiers.

4.4.4. Decision Tree C4.5 (DT)

Decision trees are supervised classification methods, which function by creating directed trees through which the data is partitioned into subgroups [251]. The resulting tree structure is comprised of nodes, starting with the root node, that is connected to several internal nodes where splits in the data are made and reaching the leaf nodes.

Internal nodes represent a decision point, where a feature is selected and used to split the data space into subspaces, in the case of binary trees the split produces two subspaces [256]. Specifically, subgroups are created, by identifying feature values that best split the data, by using a mathematical method to gauge the wellness of the split, with respect to the class value. Several methods can be used to assess the split of data from a feature, such as *chi-squared, information gain* and *cross-entropy*. Leaf nodes represent class values, which are assigned to them during training.

An important technique that is sometimes employed in the creation of decision trees is 'pruning' [246]. Through pruning, the size of the tree is regulated, as large trees tend to overfit while small trees underfit. The process involves removing sections of an overfitted decision tree that have been identified to contribute little to the classification process. By doing so, the generalization performance of the tree is improved. At the same time, pruning helps reduce the complexity of the model. Some examples of pruning methods include Error-Based Pruning and Reduced Error Pruning [246].

Being a supervised classification model, decision trees require labelled data, which are recursively separated into subgroups. During classification, a new sample is assessed at each node, creating a path on the decision tree, until reaching the leaf node that depicts the assigned class. Some popular decision tree algorithms include ID3 which uses IG as a splitting factor and C4.5 which is an upgrade form ID3 and uses gain ratio in the splitting nodes. An example of the tree-like structure of a decision tree is given in Figure 4.4.

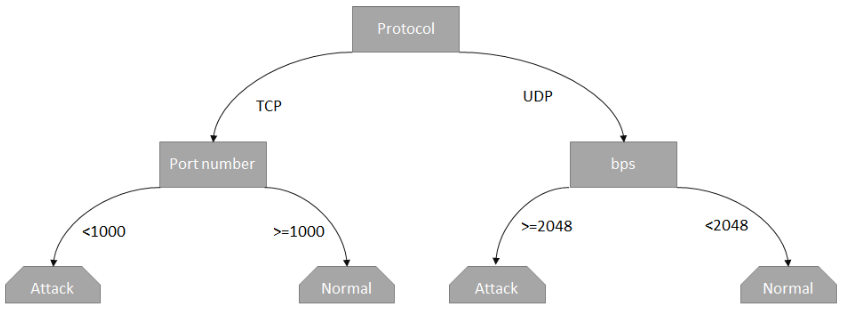


Fig. 4.4: Decision Tree example

Although decision trees are simple and fast models, provide relatively good accuracy and the process of classification is easy to comprehend, they carry some disadvantage. Disadvantages include long training time, bad memory scaling when dealing with large data and increased complexity, in some cases, compared with other models [251].

4.4.5. Deep Belief Network (DBN)

Deep Belief Networks are a class of generative deep learning and neural network models. They are comprised of a visible and a hidden segment, with the visible segment being the initial input layer of the model, and the hidden segment comprised of stacked networks like autoencoders [257]. In some implementations, the hidden layers are comprised of stacked Restricted Boltzman Machines (RBMs), with the hidden layer of one RBM being the visible layer of the next.

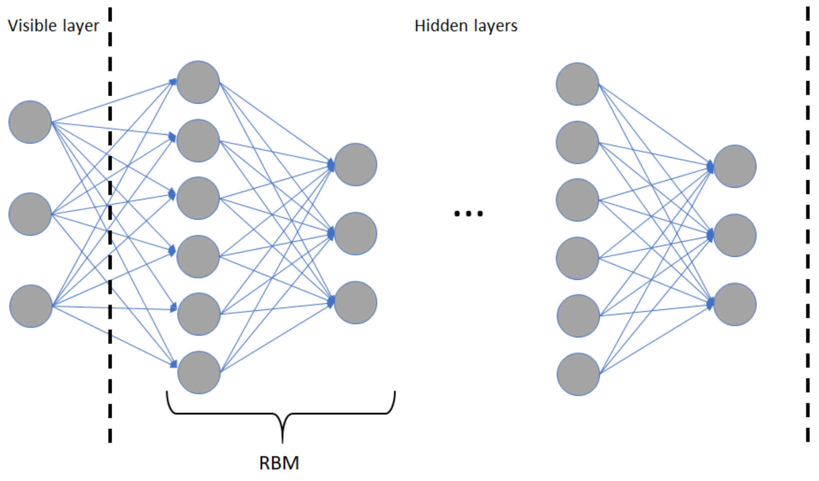


Fig. 4.5: DBN example

Training is conducted greedily, in a layer-by-layer method, with each RBM in the network trained at a time. Although primarily an unsupervised method, DBNs can be used in supervised classification. First, the DBN is trained in an unsupervised manner, with its output being a reduced representation of the original feature space and then, by stacking a classifier at the end of the DBN, the combined network is trained for classification [258].

In this sense, DBNs are used to extract features from the original feature space, which are fewer in number than the original and carry more information, a process called feature extraction. In this chapter, we used python code [259] to train the model for classification through DBNs and compare it with other classifiers. This implementation stacks a softmax linear classifier at the end of DBN to train the model for classification.

Unsupervised ML algorithms can learn from unlabeled data, utilizing similarity and distance metrics to create groups of similar data. The two unsupervised clustering algorithms used in this chapter are discussed as follows:

4.4.6. K-Means

Clustering algorithms are a family of techniques used to separate data into groups. The process is called unsupervised learning, as there is no need for the data to be tagged (class values are absent). Instead, unsupervised algorithms seek patterns and similarities in data, in order to define groups called clusters [49]. The idea behind defining clusters is to maximise the similarity of data points inside a cluster while at the same time also maximise the dissimilar between data points of different clusters. Such algorithms usually require a user-provided "K" number which indicates the number of clusters that need to be created.

There are two main types of clustering techniques based on the clusters that are produced, partitional and hierarchical. Partitional separates the data into subgroups which do not overlap, which means each data point belongs to exactly one cluster. Hierarchical clustering, on the other hand, groups the data into nested clusters, which form a kind of hierarchy, that can be represented by a tree structure.

K-Means is a partitional clustering algorithm that uses centroids to define clusters [260]. Membership to a cluster is determined by the relative distance of a data point to the centroids of the clusters. Specifically, the data point is assigned to the cluster, the centroid of which is closest to that data point compared to the other clusters' centroids. An example of assigning a new data point to a cluster, where the number of clusters is three in Figure 4.6.

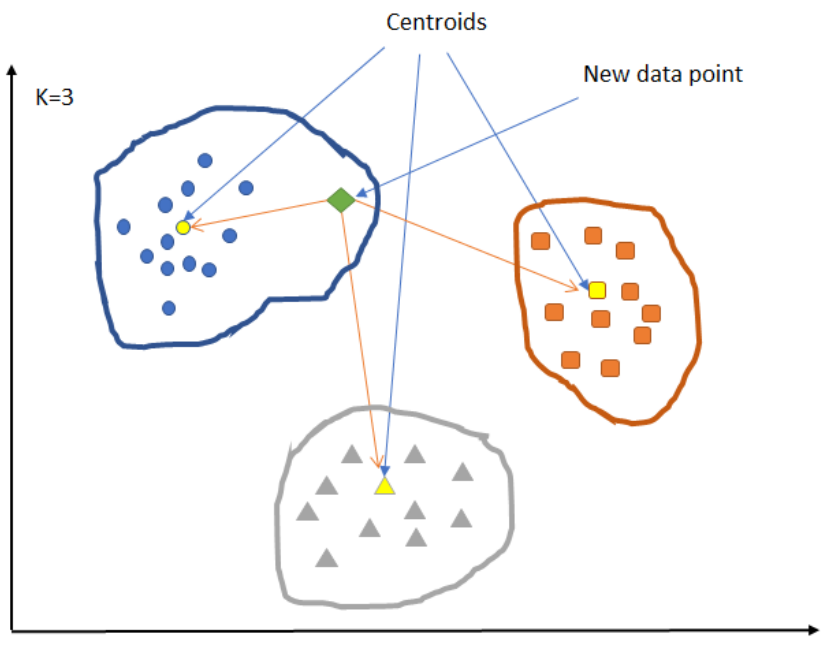


Fig. 4.6: K-Means example

At the beginning of the algorithm, the centroids are often chosen randomly. Distances are calculated and the first clusters are created. The process repeats, with the centroid of each class, recalculated at the beginning of the algorithm's iteration [260]. Membership of a data point "X" is given in Equation 4.4.6, where c_i is the centroid of class C_i and the assigned class being the class for which its centroid has the minimum distance from the data point "X". The centroids, as the name suggests, are produced by the mean of the data points in each cluster. For calculating the distance, a number of methods can be used, one of which is Euclidean distance, given in Equation 4.4.5.

$$D(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_i - y_i)^2} \tag{4.4.5}$$

$$Memb(X) = C_i : min(D(X, c_i)), c_i \in C_i \tag{4.4.6}$$

The algorithm will execute until a user-provided number of iterations has passed or until the change in centroids is very small. In this chapter, we utilized a

Weka implementation of K-Means, which ran on all three datasets and we evaluated the resulting clusters by using the class feature to produce confusion matrices. Because we have performed binary classification with the other models, we set the "K" value to two, indicating that two clusters should be constructed.

4.4.7. Expectation-Maximization (EM)

EM is a clustering algorithm used to find maximum-likelihood estimates in unlabeled data. It is primarily used to determine several parameters of the data, like mean and standard deviation, that best describes the distributions in the mixed data, separating it into clusters. In other words, during the clustering process, the EM algorithm seeks to estimate characteristics such as mean and standard deviation for each cluster, so that the likelihood of a data point's membership to a cluster is maximized [261].

Both EM and K-Means are similar, in that their process repeatedly alters their clusters in an iterative fashion, although their underline techniques differ. Instead of calculating distances of data points and cluster centroids, seeking to maximize them, EM computes probability estimates of a data point belonging to a cluster, based on estimates of the clusters' distributions. The algorithm works on two repeating steps, first, it estimates the possibility of membership of a data point to a cluster, based on some parameters, then it re-computes these parameters based on the previous step's possibilities results [262]. Datapoints are then assigned to clusters, based on their membership possibility. Initial parameters are randomized, similar to K-Means. Equations 4.4.7 and 4.4.8 give the Gaussian probability for data point x_i given class C_j where 'j' is either '0' or '1' if there are only two clusters and the Bayes-derived membership probability for that data point.

$$P(x_i/C_j) = (1/\sqrt{2\pi\sigma^2_{C_j}})e^{-(x_i-\mu_{C_j})^2/(2\sigma_{C_j})} \tag{4.4.7}$$

$$P(C_1/x_i) = P(x_i/C_1)P(C_1)/(P(x_i/C_1)P(C_1) + P(x_i/C_0)P(C_0)) \tag{4.4.8}$$

We used Weka's EM implementation, which allows for the number of clusters to be either user-defined or automatically generated by the algorithm. The process of automatically determining the number of clusters involves cross-validation and starts by assigning 1 cluster. The EM algorithm is executed in a 10-fold cross-validation scheme. After averaging the likelihood over the 10 folds, if there is an increase, then the number of clusters is increased by one, and the process is repeated.

During our experiments, we used the default parameters for the classification and clustering algorithms. These options can be seen in Table 4.1. It should be noted here that for the Artificial Neural Network we chose the VotedPerceptron algorithm, for the Association Rule Mining the OneR and for the Decision Tree the C4.5 (J48) algorithms.

4.4.8. Evaluation metrics

In order to compare the displayed performance of the models that were selected, and discussed in the previous subsection, multiple confusion matrices [263] were constructed. Table 4.2 introduces the general structure of a confusion matrix. In short, a confusion matrix depicts all possible results of a binary classification. Thus, a confusion matrix can depict four results, a True Positive (TP), where an attack was detected, a True Negative (TN), where a normal flow was detected, a False Positive (FP), where a normal flow was misclassified as an attack, and a False Negative (FN), where an attack was misclassified as normal traffic.

A combination of the four values TP, TN, FP, FN results in the creation of four metrics, namely Accuracy, True Positive Rate, False Positive Rate and False Alarm Rate, which are used in this chapter for evaluation purposes. These four metrics are defined as follows:

- **Accuracy** - represents the fraction of correctly classified records, in relation to the total number of records that were processed. The calculation of Accuracy (Overall Success Rate) is OSR= (TN+TP)/(TP+FP+TN+FN).
- **True Positive Rate (TPR)** - represents the rate of correctly classified as positive records from all positive records in the dataset. The calculation of the True Positive Rate is TPR=TP/(TP+FN).
- **False Positive Rate (FPR)** - represents the rate of incorrectly classified as negative records from all negative records in the dataset. The calculation of the False Positive Rate is FPR=FP/(FP+TN).

Table 4.1: Default parameters for the clustering and classificaiton algorithms

Artificial Neural Network	Batch size	100
	Exponent	1
	Max K	10000
	Decilam Places	2
	Iterations	1
	Random seed	1
Naive Bayes	Batch size	100
	Decimal Places	2
	Kernel Estimator	False
	Supervised Discretization	False
Association Rule Mining	Batch size	100
	Minimum Bucket Size	6
	Decimal Places	2
Decision Tree	Batch Size	100
	Binary Splits	False
	Collapse Tree	True
	Confidence Factor	0.25
	Minimum Instances per Leaf	2
	Decimal Places	2
	Folds	3
	Random seed	1
	Subtree Raising	True
	Unpruned	False
Deep Belief Network	hidden__layers__structure	[256,256]
	learning__rate__rbm	0.05
	learning__rate	0.1
	n__epochs__rbm	1
	n__iter__backprop	1
	batch__size	1024
	activation__function	ReLu
K-Means	Iterations	500
	Clusters	2
	Random seed	10
	Initialization Method	Random
	Distance Function	Euclidean
Expectation Maximization	Iterations	100
	Maximum Number of Clusters	2
	Minumum Log Likelihood Improvement of Cross Validation	1.0E-6
	Minumum Log Likelihood Improvement Iterating	1.0E-6
	Minimum standard deviation	1.0E-6
	Number of Clusters	2
	Number of Execution Slots (Threads)	6
	Number of Folds	10
	Number of K-Means Runs	10
	Random seed	100

Table 4.2: Confusion matrix structure

	Actual Negative	Actual Positive
Predicted Negative	TN	FN
Predicted Positive	FP	TP

- **False Alarm Rate (FAR)** - represents the fraction of incorrectly classified records, in relation to the total number of records that were processed. The calculation of the False Alarm Rate is FAR = (FP+FN) /(FP+FN+TP+TN).

4.5. Experimental Results

4.5.1. Selected datasets and feature selection

For the purpose of comparing the seven Machine Learning algorithms That were discussed in previous sections, three datasets were selected, specifically, the UNSW-NB 15 [57], the NSL-KDD [264] and the Bot-IoT [265]. The UNSW-NB 15 was selected, due to its contemporary nature, as it incorporates realistic normal traffic in addition to Botnet-related attacks, and has seen wide use in current research. The generation process of UNSW-NB 15 involved the utilisation of the IXIA PerfectStorm tool, which was tasked with producing both legitimate normal traffic, as-well-as attack traffic. The attacks represented in the dataset are grouped into 9 categories, namely Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. In its default form, the UNSW-NB 15 dataset is made up of 47 independent features, 2 class features, and a total of 257,673 records acquired by combining the training and testing sets. For the purpose of training the models, the features of the UNSW-NB15 dataset were ranked based on Information Gain Ranking Filter (IG) and the 10-best were selected, as depicted in table 4.3.

The attacks present in the UNSW-NB 15 were briefly described by Moustafa et al. [266] and include several categories, namely Fuzzers, Analysis, Backdoor, DoS, Exploit, Generic, Reconnaissance, Shellcode and Worm. Attackers use Fuzzers in order to discover weaknesses in a system's security through large volumes of randomly generated data. A Backdoor is utilised after a system's security has been compromised, giving an attacker persistent access to an already compromised machine. Techniques that attempt to disrupt the normal operation of a system and render its services unreachable are called Denial of Service (DoS) attacks.

Exploits are attacks that take advantage of (possibly) previously unknown bugs in code, rendering a system vulnerable. Attacks that are classified as Generic, target cryptographic block-cyphers and attempt to produce collisions. Prior to any attack, Reconnaissance is performed, during which information about a potential future target is gathered through probes. Shellcode attacks make use of specialised code that is injected to a live application and can lead to a remote machine being taken over by the attacker. Finally, Worms are a group of malware that spread between hosts of a network and replicate themselves.

Table 4.3: UNSW-NB15 Features selected with Information Gain

Ranking	Feature name	Feature description
0.654	sbytes	Source to destination transaction bytes
0.491	dbytes	Destination to source transaction bytes
0.477	smean	Mean packet size transmitted by source
0.464	sload	Source bits per second
0.454	ct_state_ttl	Number for each connection state according to specific range of values for source/destination time to live
0.444	sttl	Source to destination time to live value
0.439	dttl	Destination to source time to live value
0.429	rate	Connection rate
0.409	dur	Record total duration
0.406	dmean	Mean packet size transmitted by destination

The same machine learning models were used by [266] for a similar evaluation of performance on the NSL-KDD [264], a dataset derived from KDD99 and remains one of the most widely used datasets for security research to this day. The NSL-KDD dataset, although outdated, has been used to validate security solutions by many experts [267]. We chose NSL-KDD instead of KDD99, as the original dataset had duplicate entries, and required some processing before it was ready to be used for training and testing purposes. It is comprised of seven weeks of simulated network traffic that represents a military network, with several attacks organized in four categories: DoS where an attacker attempts to prevent legitimate users from accessing services, Remote to Local where a remote attacker attempts to gain access to the internal network, User to Root where an attacker has limited access to a machine in the internal network and attempts to gain root access and Probe where an attacker gathers information about a target in the network. Similarly to the process for UNSW-NB15, we extracted 10 features through IG Ranking Filter from the NSL-KDD dataset as depicted in Table 4.4.

Finally, the Bot-IoT dataset, which we generated in our previous work [265], is also utilized in this chapter. In short, we generated both normal and attack traffic, incorporating normal traffic IoT simulations, with a number of attacks organized in attack types and attack subtypes. Some of the attack types that were generated, include DoS, DDoS, Information theft, Probing. For the work depicted in this paper, we extracted 10 features through IG Ranking Filter from the 5% full-feature version of the Bot-IoT dataset as depicted in Table 4.5, similar to the process used for the UNSW-NB15 and KDD99 datasets.

Table 4.4: NSL-KDD Features selected with Information Gain

Ranking	Feature name	Feature description
0.701	src_bytes	Number of bytes transferred from source to destination in a single connection
0.571	dst_bytes	Number of bytes transferred from destination to source in a single connection
0.428	service	Network service
0.324	flag	Flag of the connection
0.288	diff_srv_rate	Percentage of connections to different services, among the connections aggregated in count
0.275	dst_host_srv_count	Number of connections having the same port number
0.268	same_srv_rate	Percentage of connections to the same service, among the connections aggregated in count
0.244	dst_host_same_srv_rate	Percentage of connections to same services, among the connections aggregated in dst_host_count
0.241	dst_host_diff_srv_rate	Percentage of connections to different services, among the connections aggregated in dst_host_count
0.216	serror_rate	Percentage of connections that have flag s0, s1, s2 or s3, from the connections aggregated in count

Table 4.5: Bot-IoT Features selected with Information Gain

Ranking	Feature name	Feature description
0.00172	TnBPSrcIP	Total Number of bytes per source IP
0.0017	TnP_PerProto	Total Number of packets per protocol
0.00161	TnBPDstIP	Total Number of packets per Destination IP
0.00147	AR_P_Proto_P_SrcIP	Average rate per protocol per Source IP.(calculated by pkts/dur)
0.00146	sbytes	Source-to-destination byte count
0.00141	AR_P_Proto_P_Dport	Average rate per protocol per dport
0.00134	TnP_Per_Dport	Total Number of packets per dport
0.0013	TnP_PDstIP	Total Number of packets per Destination IP
0.0012	TnP_PSrcIP	Total Number of packets per source IP
0.00116	bytes	Totan number of bytes in transaction

4.5.2. Experimental results

The confusion matrices of the seven classification and clustering algorithms are listed in Tables 4.6 - 4.11 on the three datasets: UNSW-NB15, NSL-KDD and Bot-IoT. Weka was used to train and test the seven algorithms, with the parameters set to Weka's default values, which are given in Table 4.1. To better evaluate the performance of the tested models, 10-fold cross-validation was used.

Table 4.6: Combined confusion matrix for Naive Bayes

Naive Bayes	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	84101	61380
	Attack	8899	103293
NSL-KDD	Normal	14214	7453
	Attack	1387	13988
Bot-IoT	Normal	365	961
	Attack	112	3667084

Table 4.7: Combined confusion matrix for Association Rule Mining

ARM	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	31785	12675
	Attack	10894	108654
NSL-KDD	Normal	14039	783
	Attack	1562	20658
Bot-IoT	Normal	399	20
	Attack	78	3668025

Table 4.8: Combined confusion matrix for Artificial Neural Network

ANN	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	2719	2562
	Attack	90281	162111
NSL-KDD	Normal	8819	16531
	Attack	6782	4910
Bot-IoT	Normal	206	9634
	Attack	271	3658411

Table 4.9: Combined confusion matrix for Decision Tree

DT	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	84607	9058
	Attack	8393	155615
NSL-KDD	Normal	15341	315
	Attack	260	21126
Bot-IoT	Normal	461	10
	Attack	16	3668035

Table 4.10: Combined confusion matrix for Deep Belief Network

DT	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	0	0
	Attack	93000	164673
NSL-KDD	Normal	0	0
	Attack	15601	21441
Bot-IoT	Normal	0	0
	Attack	477	3668045

Table 4.11: Combined confusion matrix for K-Means

K-Means	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	90422	162004
	Attack	2578	2669
NSL-KDD	Normal	658	13138
	Attack	14943	8303
Bot-IoT	Normal	113	2833653
	Attack	364	834392

Table 4.12: Combined confusion matrix for Estimation Maximization

K-Means	Predicted\Actual	Normal	Attack
UNSW-NB15	Normal	23718	56205
	Attack	69282	108468
NSL-KDD	Normal	13729	7534
	Attack	1872	13907
Bot-IoT	Normal	366	1842
	Attack	111	3666203

In Table 4.13 the accuracy and false alarm rate for the classification and clustering algorithms trained on all three datasets are given.

Table 4.13: Combined metrics for seven machine learning algorithms

	Accuracy	TPR	FPR	FAR
	Naïve Bayes			
UNSW-NB15	72.73%	62.73%	9.57%	27.27%
NSL-KDD	76.14%	65.24%	8.89%	23.86%
Bot-IoT	99.97%	99.97%	23.48%	2.925e-2 %
	ARM			
UNSW-NB15	85.63%	89.55%	25.53%	14.37%
NSL-KDD	93.67%	96.35%	10.01%	6.33%
Bot-IoT	99.99%	99.99%	16.35%	2.671e-3 %
	ANN			
UNSW-NB15	63.97%	98.44%	97.07%	36.03%
NSL-KDD	37.06%	22.9%	43.47%	62.94%
Bot-IoT	99.73%	99.74%	56.81%	0.27%
	DT			
UNSW-NB15	93.23%	94.5%	9.025%	6.77%
NSL-KDD	98.45%	98.53%	1.67%	1.55%
Bot-IoT	99.99%	99.99%	3.35%	7.087e-4 %
	DBN			
UNSW-NB15	63.9%	100%	100%	36.09%
NSL-KDD	57.88%	100%	100%	42.11%
Bot-IoT	99.98%	100%	100%	0.013%
	K-Means			
UNSW-NB15	36.13%	1.62%	2.77%	63.87%
NSL-KDD	24.19%	38.72%	95.78%	75.81%
Bot-IoT	22.75%	22.74%	76.31%	77.25%
	EM			
UNSW-NB15	51.29%	65.86%	74.49%	48.70%
NSL-KDD	74.60%	64.86%	11.99%	25.39%
Bot-IoT	99.94%	99.94%	23.27%	0.05%

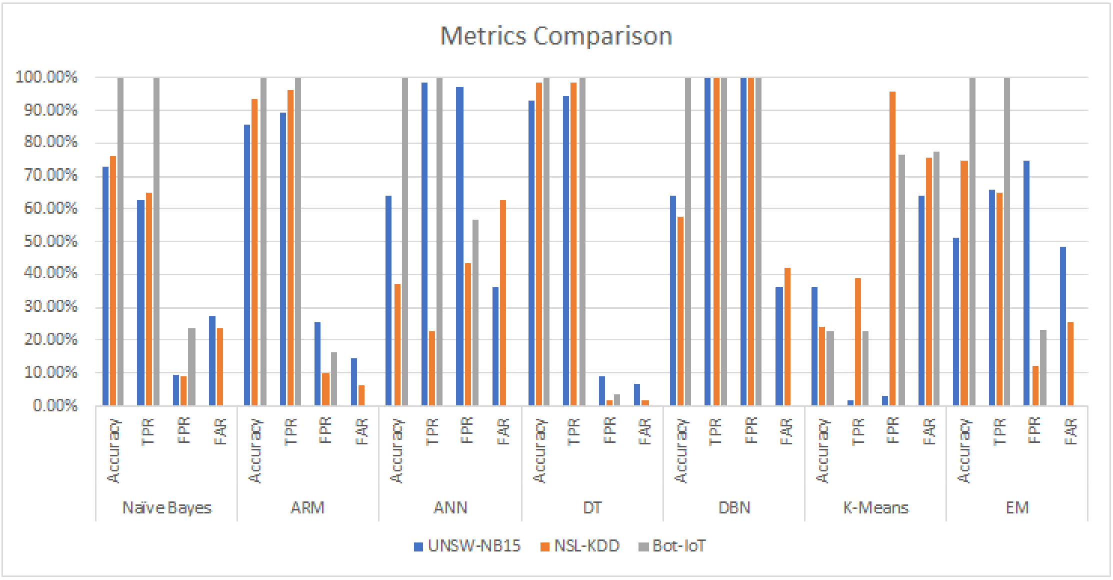
4.6. Discussion of results

Our experiments with the UNSW-NB15 demonstrated that the best model for identifying Botnet activities in captured network traffic was the Decision Tree (DT) classifier, and more precisely, the C4.5 algorithm. The C4.5 implementation utilizes Information Gain during the tree construction, in order to identify the feature subset that produces the best splits in the data according to the classification feature. The obtained results indicated that the DT displayed the best overall performance from the tested algorithms, with an accuracy of 93.23% and FAR at 6.7% which was the lowest value for that metric. The next best classifier was ARM, with its accuracy being at 86% and FAR more than twice the value displayed by the DT. The probability-reliant Naïve Bayes classifier displayed the next best performance, with its accuracy being 20% lower and its false alarm 21% higher than the DT's. The least accurate model from the five supervised algorithms that were tested, was the Deep Belief Network. Although it displayed the highest TPR at 100% it also had the highest FPR at 100%, with the ANN having the second-worst performance, as its accuracy was 30% lower and it is FAR was 30% higher than the DT's respectively. The K-Mean algorithm, one of the two unsupervised algorithms that we tested was the least overall accurate model, nearly 60% less accurate than DT while also having the smallest TPR and highest FAR.

Results for the NSL-KDD dataset indicate that the Decision Tree model again displayed the highest accuracy at 98.45% in addition to having the highest TPR value out of the seven models. The ARM classifier was the second-best classifier, similar to the order for UNSW-NB15, although its FPR was nearly three times that of the DT. The Naïve Bayes classifier was third, with 22% less accuracy than the DT and approximately five times more FPR. Both the ANN and the K-means algorithms were the least accurate, with 37% and 24% accuracies respectively. The ANN and DBN displayed the most errors out of the five supervised algorithms, with the former having the highest FAR and the latter the highest FPR. K-Means and DBN both produced the largest errors out of the seven models that were tested, with FPR at 100% for the latter and FAR at 76% for the former.

Similar to the results of the other two datasets, results for the Bot-IoT indicate that the DT and the ARM outperformed the other three algorithms, displaying the highest accuracy and TPR at 99.99%. For the supervised algorithms, the highest FPR was produced by the ANN at more than 56%, although this error for the K-Means algorithm was 20% higher. The smallest FPR was given by the DT at 3%, with the next best value being that of ARM at just under five times that of the DT. The K-Means algorithm again performed poorly, having the smallest accuracy and TPR at just under 23% while also having the largest FAR out of all five algorithms at 77%.

Fig. 4.7: Comparison of metrics for the three datasets



Overall, as can be seen in Table 4.7, models trained on the Bot-IoT displayed the highest accuracy and TPR. Out of the seven algorithms that were tested, K-Means displayed the worst performance, with the lowest accuracy, while out of the five supervised algorithms, the DBN was outperformed. Regarding the DBN performance, although it displayed the highest TPR for all the dataset, it achieved it by misclassifying all the negative results, thus having the highest FPR out of all the tested models as well. This can be explained by the choice of default/unoptimized hyperparameters, as deep models often perform well with large quantities of data. Regarding the two unsupervised algorithms, as can be seen in the diagram and the generated metrics, K-Means was outperformed by the EM clustering algorithm, with the latter displaying a higher accuracy in all three datasets, higher TPR and lower FAR. The DT consistently was the most accurate at detecting botnet activities from the datasets, with the lowest error given by the FAR. It can be observed that machine learning can be applied as network forensics models and they are capable to discover botnet events.

4.7. Conclusion

In this chapter, we have used various machine learning models to examine how they would be used for developing network forensics that can identify botnet events from IoT networks. Five supervised machine learning algorithms: ARM, ANN, Naive Bayes, Decision Tree, Deep Belief Network and two unsupervised machine learning algorithms K-Means, Expectation-Maximization were trained in a 10-fold cross-validation scheme on the datasets of Bot-IoT, UNSW-NB15 and NSL-KDD. By comparing their results, the quality of the Bot-IoT is better than others, as it has a variety of botnet events with significant data features. Although the classifiers displayed high accuracy, it was noted that their false positive rate was significant, with Decision Trees displaying the smallest value at 3.35% and the highest by DBN at 100%. The experimental results revealed that they can be used as a better alternative than existing rule-based network forensics tools as they do not need updating rules regularly. In the future, this work will apply network forensics models in a real-world IoT network and determine their scalability for handling heterogeneous data sources.

Chapter 5

Development of a Novel Network Forensic Framework based on Deep Learning for the Internet of Things: The Particle Deep Framework

5.1. Overview

With the proliferation of the Internet of Things (IoT) systems, IoT device-related IP addresses are widely linked to the Internet to offer daily services and tasks to end-users and organisations. The IoT industry has experienced accelerated growth over the last few years, with projections supporting the continuation of this increase [5]. In 2018, approximately 7 billion IoT devices were connected to the Internet, while in 2019 that number is expected to double to 14 billion devices. One of the most popular applications for IoT in terms of deployed devices is the smart home sector which had 663 million devices in effect in 2017. Applications for smart homes include among others, smart lights, fridges, ovens, thermostats and locks. Another application of IoT, on a larger scale, has been planned for several European countries, and it is called the "smart city" [268, 269]. Industrial, agricultural and health applications are also on the rise with automation, cost efficiency and precision being the most integral contributors to this trend. Some examples of devices from the healthcare sector include patient monitors, energy meters and imaging-related devices (X-ray machines) [5].[1]

Even though IoT devices are preferred over conventional devices and systems, such deployments remain quite vulnerable to several attacking techniques taking advantage of both well-known and new attack vectors [270][28][9]. In the 2018

[1]The work presented in this chapter has been submitted in:
Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, A New Network Forensic Framework based on Deep Learning for Internet of Things Networks: A Particle Deep Framework, Future Generation Computer Systems

Symantec report of Internet security threats [9], the total number of attacks targeting IoT devices for 2018 exceeded 57,000, with more than 5,000 attacks being recorded each month. Attackers perform various hacking techniques, for example, Denial of Service (DoS), Distributed DoS (DDoS), ransomware and other botnets attacks, for exposing IoT systems and their networks. Hackers execute several hacking scenarios to compromise vulnerable, un-patched, un-updated and/or unencrypted IoT devices to achieve their motives, such as corrupting IoT resources, stealing sensitive information that IoT devices often store or even use the compromised devices as infection vectors. Hackers sometimes seek to compromise the physical security of smart homes by hacking smart locks and garage doors [14].

Defending against such cyber-attacks is difficult, as there exist no commonly accepted standards for designing IoT devices [270]. This means that in an IoT deployment multiple protocols, such as MQTT, Zigbee and LoRa, could interact increasing complexity and heterogeneity [17, 26, 271]. Furthermore, new attacks which rely on zero-day exploits are often preferred by attackers, as they are not easily detected by most security countermeasures. Due to the heterogeneous norm of IoT deployments, developing an efficient network forensic solution demands depth-analysis for tracing and detecting attack vectors [164] [270]. Typically, the network forensic process is segmented into several distinct phases, whereby each phase defines the necessary preparation, analysis and actions of investigation [97]. These phases are identification, collection, preservation, examination, analysis and presentation. The first three stages, define the forensic analyst's access to the crime scene and their activities. Initially, sources of potential evidence have to be identified, after which data has to be collected in a way that enables the preservation of the data and the chain of custody. In the next two stages, the collected data is processed, so that relevant evidence can be located, after which the evidence is analysed to make inferences about the cybercrime. Finally, the results are organised and further elaborated upon, so that they can be used in a court of law.

Various forensic frameworks have been proposed to provide solutions to the acquisition problem but they do not consider the entire phases of an investigation [17] [18] [20] [19]. Most of them rely on a public ledger scheme, where diagnostic and communication data are shared between multiple entities, such as the police and insurance companies. The benefit of such a scheme would be that during an investigation, all relevant information could be readily available to the forensic experts, while its integrity would be guaranteed via digital signatures. However, most of the network forensic frameworks focus on data acquisition rather than considering the entire forensic process. These frameworks introduce some disadvantages such as the violation of privacy, as a user's information is distributed between the stakeholders, and the added complexity that these frameworks require. Furthermore, the frameworks focus on the preservation and collection phases of the investigation.

To investigate network-related incidents, several types of files are studied for tracing attack vectors that expose IoT systems. The easiest source of traces to access, which is often preferred, is traffic collection where network packets are recorded and stored in full packet capture files. By examining the files, an investigator can determine if an attack occurred and find all metadata related to attack traces such as timestamp and source of attack. There are two approaches: deep packet inspection and network flow analysis, to process and examine the files and discover attack behaviours [270]. Deep packet inspection focuses on the payload of the packet, allowing for an in-depth analysis of the traffic that is captured, which may be more accurate at detecting certain attacks although it may face challenges, like the payload being encrypted, a common occurrence in current networks, it requires extensive storage and accessing the payload may violate privacy laws. On the other hand, network flow analysis utilises a summarisation of the network traffic, where mainly statistical features are extracted such as connection speed, exchanged bits and timing data, in order to produce results. For the requirements of the research presented in this paper, network flow is preferred, as we suggest in this work [270].

A key characteristic of IoT devices is that they are constantly active. As such, performing network flow collection from an IoT's network results in excessive amounts of data. In order to perform analysis on the collected data, automated mechanisms are often employed to eliminate human error, with one such popular automated method being the utilization of deep learning. Through deep learning models, an investigator is capable of rapidly detecting patterns in the network data (packets, network flows) which indicate the occurrence of an attack [158, 159, 272, 273]. However, for such a deep learning model to be used, it first needs to be trained. Training a model requires the utilization of data and the selection of values for the model's hyperparameters. Although both data and hyperparameters are important for the training phase, hyperparameter optimisation, referring to the selection of optimal hyperparameter values, is crucial as it defines the abstract structure and training conditions of the model [274, 275]. Considerable work has appeared over the years in the field of hyperparameter optimisation [274, 276, 277, 278], with the trend being a shift from the time consuming manual selection that may not yield optimal results, to more automated processes. Regardless, no single optimisation method has been accepted as the preferred method by the research community, and as such, work is ongoing. It is necessary to develop network forensic mechanisms based optimisation to timely investigate security incidents in IoT networks [270, 279, 280].

The main contributions of this chapter are as follows:

- We propose a new network forensic framework, named Particle Deep Framework (PDF), based on optimisation and deep learning.
- We use an optimisation method based on Particle Swarm Optimisation(PSO) to select the hyperparameters of the Deep Neural Network (DNN).
- We compare the performance of the produced DNN with other deep learning models. For the evaluations and comparisons, we used the Bot-IoT [265] and the UNSW-NB15 [57].

Chapter 5 is structured as follows. Section 2 includes related research in the application of deep neural networks to network forensics. Section 3 presents an overview of our proposed framework. In Section 4 we discuss the PSO algorithm and its use for hyperparameter tuning. In Section 5 we discuss the proposed PDF in detail, followed by its architectural design in Section 6. In Section 7 the results of training the PDF is given and Section 8 presents a discussion of produced results. Section 9 presents a comparison of the experimental results of the PDF with other machine learning-based models. Section 10 gives the attack identification statistics for the two datasets. Finally, Section 11 includes a comparison of the PDF with traditional IDS, followed by the concluding remarks of this chapter in Section 12.

5.2. Background and Related Work

5.2.1. Digital Forensic Frameworks for IoT and smart systems

Digital forensics is a collection of methodologies that have been produced and refined through the scientific community, tasked with the secure collection of traces from a crime scene, their examination, analysis and presentation of identified evidence which can help identify the malicious actors, their methods and motives, during an investigation of a security incident, often leading to legal action [105]. Over the years, and due to technological development, digital forensics has been refined into several subcategories, each specializing with incidents in different environments, namely: cloud forensics, network forensics, IoT forensics, mobile forensics, memory forensics, data forensics [281].

For the purposes of this chapter, we focus on network forensics. Network forensics mechanisms focus on security incidents that occur in networks, commonly using logs and captured packets to detect intruders and malicious acts. Experts who perform network forensic actions, make use of various tools for the correct collection and storage of evidence before analysis takes place. However, there is no single best process to follow during an investigation, giving rise to numerous digital forensic frameworks. In essence, a digital forensic framework dictates the appropriate steps to be taken by a professional during an investigation. It segments the investigation into distinct and autonomous phases that propose specific techniques and technologies for that phase. As previously mentioned in Section I, most forensic frameworks proposed for a smart home crime scene focus primarily on the acquisition phase of the investigation. Multiple frameworks have been proposed [18, 20, 282]. However, no framework has been singled out as the preferred choice by professionals, since standardization is lacking, and varying circumstances require different approaches and tools [270, 283, 284].

Many researchers have developed forensic frameworks for the IoT [17] [18] [20] [19]. For instance, Hossain et al. [20] and Hossain et al. [18], proposed Probe-IoT and FIF-IoT, respectively, two models which handle the acquisition of evidence from IoT devices in a forensically sound way for maintaining integrity and chain of custody, without violating the user's privacy. Meffret et al. [17] proposed an FSAIoT framework for the collection of state data from IoT devices. Following that, Cebe et al [19] developed a Block4Forensic acquisition model designed for vehicular data collection. Both Probe-IoT, FIF-IoT and Block4Forensic base their implementations on the blockchain scheme. For Probe-IoT and FIF-IoT, a distributed public ledger was established, with multiple stakeholders maintaining copies of the produced ledger, while Block4Forensic used a fragmented ledger to reduce storage requirements. In all of the aforementioned cases, the information that is stored in the blockchain includes diagnostics about IoT devices and interaction between devices and other network entities. In contrast, FSAIoT uses centralised controllers in a local network to monitor and collect states and data transactions from the devices.

A significant portion of the literature regarding the development of new forensic frameworks for IoT and smart systems was based on the concept of distributed digital blockchains [20, 20, 282]. Hossain et al. [20] proposed Probe-IoT, Le et al. [282] BIFF and Hossain et al. [18] the FIF-IoT. These frameworks have a lot of similarities. To begin with, they are all designed around the concept of a distributed digital blockchain system being maintained by several relevant stakeholders, including but not limited to: Law enforcement, insurance companies and the manufacturer of IoT devices. Pre-defined roles dictate access rights for the stakeholders, while digital signatures ensure non-repudiation of events. To enable the use of digital signatures, some trusted entity plays the role of the Certification Authority, maintaining the public-keys that might be hacked by man-in-the-middle attacks [285].

Although the framework sometimes speeds up the investigation process, as the investigator would not need to physically attend the crime scene and collect data, it comes with some drawbacks. To begin with, they require either the introduction of dedicated devices that collect transaction data or the smart devices themselves to periodically transmit such data to on-line services that collect the transactions and incorporate them to the blockchain. This may cause increased charges either for new devices, or extra power consumption for the smart devices, with possible degradation of services due to constrained resources. Furthermore, these frameworks rely on the harmonious cooperation of multiple entities, while at the same time require (i) extra resources by Law enforcement to store the collected data, and (ii) trust by the IoT device owners that register their devices [18, 20, 282].

Babun et al. [286] developed a different acquisition model named IoTDots. Its functionality relies on first modifying the applications which control the smart devices so that logs and data can be re-directed to a remote database at runtime. A secondary module pre-processes and analyzes the gathered data to identify security incidents. The analysis is done by converting all collected data into bitmaps and through training a Markov Chain model on legitimate interactions. Although IoTDots considers collection, preservation and analysis phases of an investigation, it has some drawbacks. The framework requires modified applications to be used in order for IoTDots to function. Thus, users will be under constant surveillance with their activities being recorded, which may raise privacy concerns. Additionally, during the collection and transmission of the collected data, preserving the integrity of data is not considered. The data integrity and authenticity stage is vital and needs to be applied at the early stages of the digital investigation process as the data might be modified by malicious entities during either the collection, transmission or storage.

5.2.2. Deep Learning for tracing and discovering threat behaviours

Traditional methods of network forensics include the use of fuzzy logic [287, 288], Naïve Bayes classifiers [289], neural networks [288, 289], support vector machines [290]. Deep Learning tends to be the preferred method of choice for the task of network forensics as, although training such models is time-consuming, their execution time is low and they identify more complex patterns, outperforming other choices, especially when working with large volumes of data [34, 158, 159, 270, 291].

Deep Learning is a subsection of artificial neural networks, where the neural networks have a deep architecture that span multiple hidden layers [51, 159]. Parsing logs, network traffic and documents demand a reverse-engineered code to identify indications of an attack, which is an iterative process that humans cannot easily perform. To that effect, multiple types of machine learning models have been used to harness their discriminative capabilities [270]. Recently though, deep learning has received more attention from the research community because it learns data and its variations in-depth through multiple generative or discriminative models. In deep learning, stacking tens/thousands of hidden layers together has been shown to increase the predictive capabilities of a neural network, allowing it to identify complex patterns in the data [159, 274].

A common application of deep learning in the forensic and security research areas is attack identification in network traffic. One such example by Shone et al. [159] is the development of an intrusion detection system based on a combination of a stacked non-symmetric deep autoencoder and a random forest classifier. Initially, two autoencoders are pre-trained, and their "encoder" parts are stacked, with the last one feeding its output to the random forest classifier. Evaluation of the KDD dataset depicted an accuracy of 89.22%. Azmoodeh et al. [291] proposed a deep learning approach, based on a convolutional neural network to detect "Internet of Battlefield" malware attacks. The network was trained on eigenvectors generated from the operation code sequence graph of the disassembled code of a mobile application. It was shown that this new method outperformed previous work in the field of malware detection, with results indicating accuracy and precision of about 98%.

Yuan et al. [292] proposed DeepDefence, a DDoS detection method based on deep learning. Their approach tested a combination of a convolutional neural network with three different types of temporal-aware neural networks, namely the recurrent neural network (RNN), gated recurrent unit neural network (GRU) and long-short term memory neural network (LSTM) using the ISCX2012 dataset. Results indicated that both LSTM and GRU achieved the best performance at around 98%. Brun et al. [239] developed a dense random neural network method for the identification of attacks against IoT gateways. Two attack scenarios were considered, Denial-of-Service and Denial-of-Sleep which were represented in a custom dataset. However, the accuracy of this method was reported to be comparable to a threshold detector.

Van et al. [238] built a NIDS based on Deep Belief Networks (DBN). To determine the best approach, the researchers compared the performance of a stacked Autoencoder (SA) with a stacked Restricted Boltzman Machine (SRBM) on the KDD99 dataset. Results indicate that the SA displayed a smaller error in classification compared to the SRBM, although it required more time for training. Pektas et al. [119] proposed a deep MLP system to process network flow patterns and identify botnets, specifically the communications between C&C and bots. During pre-processing, graphs were generated from collected flows and grouped by communication endpoints which allowed them to generate new statistical features. The researchers concluded that deep learning presents acceptable accuracy for botnet identification in flow data, with the added bonus that feature selection is not necessary, as deep networks identify the best features.

Cheng et al. [45] developed D2PI, a system based on a Convolutional Neural Network (CNN) which classified collected traffic into either “malicious” or “benign “, based solely on the payload. In order to train the CNN, the payloads of packets were extracted, their lengths adjusted to a predefined length and incorporated in a matrix. Results indicated that D2PI is a promising first step towards incorporating CNNs in deep packet inspection systems. Le et al. [293] produced a deep learning classification approach for the identification of different malware samples without the need for expert knowledge. The malware samples were converted to one-dimensional image representation and then used to train several different neural network models that combined convolutional layers which processed the input, with RNN and LSTM layers. Best accuracy was achieved through the CNN bi-directional LSTM, at 98.2% with a class re-balancing step.

Alrashdi et al. [294] developed an anomaly-based NIDS, called AD-IoT, that detects compromised IoT nodes, and based their design on Random Forest and Extra tree classifiers. An evaluation was performed on the UNSW-NB15, after reducing the feature size from the original 49 to 15. Results showed that the AD-IoT displayed a promising performance, although precision for detecting attacks was the lowest metric at 79%. Homayoun et al. [295] proposed BoTShark, a botnet detection and traffic analyzer based on Autoencoders and Convolutional Neural Networks (CNN). The researchers based their work on the ISCX dataset, utilizing netflow data for training and testing their method. Two versions of the BotShark were implemented, one based on stacked Autoencoders, where feature extraction is performed prior to classification and one based on a CNN, where each record was fed to the network individually. Results indicated an accuracy above 90%, with the autoencoder performing slightly better than the CNN, due to the reduced feature-set that it used.

Deep learning has been developed to detect and trace attacks from industrial IoT systems and their networks. De La Torre et al. [296] proposed a conceptual monitoring framework. They first surveyed the existing research on forensics, with emphasis on the many types of deep packet inspection (DPI). Reaching the conclusion that, relying on purely DPI solutions would be ineffective due to encrypted network traffic and the mutability of attack patterns, the researchers proposed a software-defined network-based (SDN) monitoring system. This conceptual system dictates the use of a forensic black-box, where monitoring traffic from smart grid networks and control stations would be gathered, eliminating difficulties with the acquisition phase. This black-box can then be accessed, and deep learning models applied for the identification of attack patterns, for both forensic and future prevention purposes.

As previously discussed, most of the existing studies have focused on acquisition approaches [18, 20, 282], or modifications to controller applications [286]. The new proposed framework, PDF is a viable alternative, as it harnesses network flow data which produce results without raising privacy concerns. Furthermore, it considers the analysis and examination phases which were overlooked by many frameworks that were presented, while not requiring the introduction of new entities or alterations to existing IoT and smart systems.

5.2.3. The Particle Swarm Optimisation algorithm

Particle Swarm Optimisation (PSO) is an optimisation swarm-based algorithm originally proposed in 1995 by Eberhart and Kennedy [297]. The algorithm identifies a solution by iteratively traversing the search space, and gauging its quality through the use of an objective function. The PSO algorithm is considered to be metaheuristic, since it does not rely on any assumptions about the underline problem, and is utilised in order to detect if not optimal, then "good enough" solutions in a reasonable time [298].

PSO is often utilized to determine the value of a variable, as such, the search space that the PSO algorithm traverses, is defined as all the possible values that the variable can take. The PSO algorithm functions by spawning a population of particles, with each particle defined by their current position in the search space, the best position that the particle has observed so far, a global best position achieved by some particle in the swarm and a relative velocity[298, 299]. The particles are initialized and set to traverse the search space randomly, with an objective function used to gauge the wellness of the new position of the particle, and possibly update the local best and/or the swarm's best solution.

Since the introduction of the original PSO by Eberhart and Kennedy [297], the equation of which is given in equations 3-6, researchers have proposed variants of the algorithm, that improve its performance for certain problems or extend its usability. To begin with, altering the values of the learning factors θ_1 and θ_2 of equation 5, has a direct impact on the search pattern that the swarm focuses on as reported by Kennedy et al. [297]. For instance, by diminishing the global search (setting θ_2 very close to or equal to '0'), forces the swarm to emphasize the individual local search of each particle while having $\theta_1 = \theta_2$ causes the swarm to gravitate towards the average of the global and local best solutions.

The standard PSO (SPSO) introduced by Shi and Eberheart [300] introduced an inertial coefficient (ω), which was multiplied to the particle's previous velocity in equation 5. By having relatively large values of inertia, the particles prioritize exploration of the search space, as their previous velocity has a greater impact on their new velocity in each step. The inertial weight can be initialized in any number of ways, such as randomly [301], it can be set as a positive non-zero value [300], or be a function of time either non-linear or linear [302, 303] as given by the following gradually declining Equation:

$$\omega_t = \omega_{max} - \frac{i}{i_{max}} * \omega_{max} - \omega_{min} \tag{5.2.1}$$

Where ω_{max} and ω_{min} are pre-defined max and min weight values and i and i_max denote the current and maximum iterations of the swarm respectively. The reason for having a time-decaying inertia weight is that it causes particles to explore the search space during early iterations, and then gravitate towards local search as inertia decays.

By reviewing the OPSO and SPSO, researchers observed that some particles' velocity tended to explode. In order to counter these issues, improvements were devised, such as introducing velocity clamping and constriction factors [304, 305, 306]. In order to manage the exploration-exploitation tradeoff, the velocity of each particle is calculated and limited to a pre-defined maximum value. By selecting large velocity maximums, particles explore greater areas of the search space, while smaller values focus the search to a limited area. The constriction factor was introduced as an alternative to the inertial coefficient. To apply the constriction factor, the OPSO equation for velocity (5) is adjusted, by multiplying the constriction factor K to the new velocity, with the factor given by the Equation:

$$K = 2/|4 - \phi - \sqrt{\phi^2 - 4\phi}|, where\phi = \theta_1 + \theta_2 and\phi > 4. \tag{5.2.2}$$

Next, several PSO variants were developed in order to adjust the algorithm and enable its application in diverse problems [307]. The binary PSO [307] was developed, in order to enable the application of PSO to binary problems. Its novelty was to use the calculated velocity, produced by equation 5 with velocity constriction or clamping, as the input for a sigmoid function, and using the produced value (in the [0,1] range), to set the new position of the particle as either '0' or '1'. The fully informed PSO [308] is a variant of the SPSO, where a particle's movement is mostly affected by its neighbours. Specifically, instead of using the best position that the swarm has identified in order to update a particle's velocity, its neighbour's position is used. An altogether separate approach was taken with the cooperative PSO [309], where the problem was split, based on the dimension of the underline problem. In this variant, each swarm traverses a search space corresponding to a single dimension of the solution vector. As each swarm functions separately, special attention is given as to how the acquired partial solutions should be combined.

5.3. Overview of proposed Particle Deep Framework

We present the stages of the new network forensic framework, so-called Particle Deep Framework (PDF), based on particle swarm optimisation and deep learning for tracing attack origins and detect them from IoT networks, as depicted in Figure 5.1.

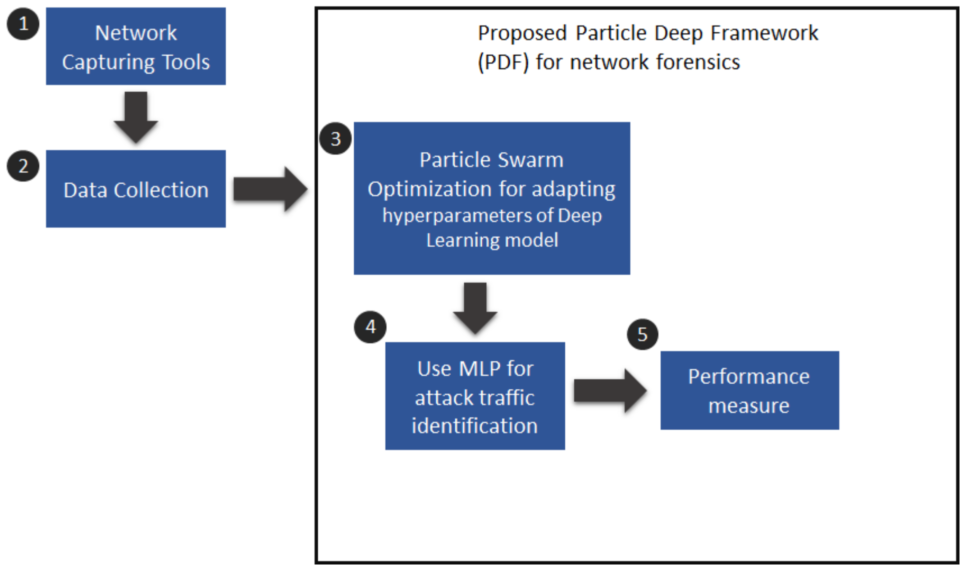

1
Network Capturing Tools
2
Data Collection
Proposed Particle Deep Framework (PDF) for network forensics
3
Particle Swarm Optimization for adapting hyperparameters of Deep Learning model
4
Use MLP for attack traffic identification
5
Performance measure

Fig. 5.1: Proposed network forensic framework using particle swarm optimisation and Multi-layer Perceptron (MLP) deep learning algorithms.

The stages of the proposed framework are separately discussed as follows.

- **Stage 1: Network capturing Tools:**

  IoT devices have been placed into a network that is under investigation. The devices have been configured in a promiscuous mode, thus enabling them to see all traffic in a local network. Network packet captures are then carried out by utilizing network capturing tools such as Wireshark [310], Tcpdump [311] and Ettercap [312]. For example, during the collection of raw pcap files for the creation of Bot-IoT, we used T-shark [65]. T-shark is a terminal-based tool, which lacks a Graphical User Interface (GUI) but otherwise has the same functionality as Wireshark. By setting the Network Interfaces Card (NIC) in promiscuous mode, we were able to use t-shark to capture packets that were generated by the machines connected to the virtual local network.

When running t-shark the command that we employed specified the network interface to use thorough the "-i" option, the filesystem path to output the collected traffic "-w" and the maximum size for each .pcap file "-b". The collected pcap files are then forwarded to the data collection stage.

- **Stage 2: Data Collection and Management Tools:**

  This is the first stage in the network investigation process, where data are gathered in a form that can be further analyzed and examined, such as the datasets of BoT-IoT and UNSW-NB15. Initially, for the purposes of preservation, an SHA-256 hashing function [313] is employed to maintain the integrity of the collected data. Through this hashing function, the produced digests of the collected files can be used post-investigation to assert that the initial data have not been compromised. The collected pcaps are then processed by data flow extraction tools like Bro [226] or Argus [191], that extract and summarize extract the network flows from the pcap files. An additional step during this stage is preprocessing, by handling missing and unuseful feature values, re-scaling and producing new features which can assist a model's training process. After filtering and cleaning datasets, the particle swarm optimisation and deep learning models are applied to discover cyber-attacks and trace their origins.

- **Stage 3: Particle Swarm optimisation (PSO) for adapting hyperparameters of Deep Learning model:**

  The PSO algorithm [298] is chosen in order to adapt the hyperparameters of the deep model as it can easily address the local-optimum problem and quickly converges to obtain best fitness values compared with other evolutionary algorithms [299, 314]. The PSO has been used to minimize/ maximize an objective function, specifically in this study, it is used to maximize the Area Under Curve (AUC) values of a deep Multi-Layer Perceptron (MLP) algorithm to obtain the best hyperparameters that will be used to train and validate the algorithm for discovering cyber attacks and identifying their origins. To be more precise, this stage uses PSO to identify the optimal hyperparameters that maximize the AUC of the deep model.

- **Stage 4: MLP deep learning for attack identification:**

  The hyperparameters that have been estimated by the PSO algorithm and the collecting data from Stage 2 are used to train and test the MLP deep learning algorithm. The MLP was adopted by seven layers (excluding the input layer), with the number of neurons being: 20, 40, 60, 80, 40, 10, 1, as the best outputs were produced in terms of detection accuracy and false alarm rate. The three hyperparameters epochs, learning rate and batch size obtained from Stage 3 are used for training and validating the MLP algorithm. Data collection of stage 2 has been split into two groups: Training, and Testing split by 80%, 20% for measuring the performance of the MLP algorithm.

- **Stage 5: Performance measure:**

  Finally, performance metrics are obtained by running the trained deep MLP model on the Testing and validating data. Common performance metrics, including accuracy, precision, recall, false-positive rate, False-negative fate and f-measure, are estimated to measure discriminative capabilities of the proposed model. The details of Stages 3-5 are explained in the following two sections.

5.4. Particle Swarm Optimisation (PSO) Algorithm for deep learning parameter estimations

Particle Swarm Optimisation is an evolutionary algorithm which was derived by observing swarms of fish and birds in nature [298]. A particle swarm is comprised of a pre-selected number of particles (P). During runtime, each particle is randomly initialized and starts propagating through the search space (v).

When a particle reaches a new position in the search space (v_{t+1}), an objective function is used to determine the quality of that position, with the function changing, depending on the problem that is optimised. Each particle is defined by a group of four vectors, its current position in the search space (x_t), its velocity (v_t), the best position identified by it (x_{lbest}) and the global best position (x_{gbest}), as declared in the following equations.

$$P = p_1, p_2, \dots, p_n, n \in \mathbb{N} \tag{5.4.1}$$

$$\forall p_n \in P, p_n = (x_t, v_t, x_{lbest}, x_{gbest}) \tag{5.4.2}$$

$$v_{t+1} = v_t + \theta_1 * rand * (x_{lbest} - x_t) + \theta_2 * rand * (x_{gbest} - x_t), rand \in [0, 1] \tag{5.4.3}$$

$$x_{t+1} = x_t + v_{t+1} \tag{5.4.4}$$

In the above equations, v_{t+1} is the updated velocity of a particle. Thus the velocity of the particle at time t+1, depends on its previous velocity v_t, the learning factors θ_1 and θ_2 which are multiplied by a random number (rand) within the $[0, 1]$ set and the distance between the local and global best solutions from the current position of the particle. The updated position x_{t+1} of the particle is calculated by adding the new velocity (v_{t+1}) to the previous position (x_t). In the following algorithm, a typical iteration of a maximizing PSO is given [298].

Algorithm 5.1 Particle Swarm Optimisation maximization algorithm

```
P ← construct_particles(n_particles)  ∀ p ∈ P, $p.X_{lbest} = p.x_0, p.X_{gbest} = -\infty$  epochs
← load_epochs()  e←0  while $e <epochs$ do
    foreach  $p \in P$ do
        $v_{t+1} = v_t + \theta_1 * rand * (x_{lbest} - x_t) + \theta_2 * rand * (x_{gbest} - x_t), rand \in [0,1]$  $x_{t+1} =$
        $x_t + v_{t+1}$  if $x_{t+1} > x_{lbest}$ then
            $x_{lbest} = x_{t+1}$
        end
        if $x_{t+1} > x_{gbest}$ then
            $x_{gbest} = x_{t+1}$
        end
    end
end
return $P.global\_best()$
```

Algorithm 5.1 is used to get the optimal hyperparameters of the MLP deep learning model to ensure the highest detection accuracy of detecting and tracing attack vectors. The particle swarm algorithm is initialized and tasked with maximizing the AUC values of the deep learning model, by seeking the best values of the hyperparameters: batch size, epochs, and learning rate, in their respective search spaces.

There are multiple reasons for using the PSO algorithm instead of another metaheuristic for the purpose of hyperparameter selection. To begin with, PSO is a simple algorithm to implement and its inner workings can be understood easily [315]. PSO does not guarantee an optimal solution however, it has been shown that it can produce satisfactory results in reasonable time [316]. Furthermore, although PSO tends to converge faster than other metaheuristic algorithms [317], it can be further sped up through parallelization of some of its parts, such as calculating the objective function's value. Finally, since our research has indicated that PSO has not been used for hyperparameter optimisation, this work provides

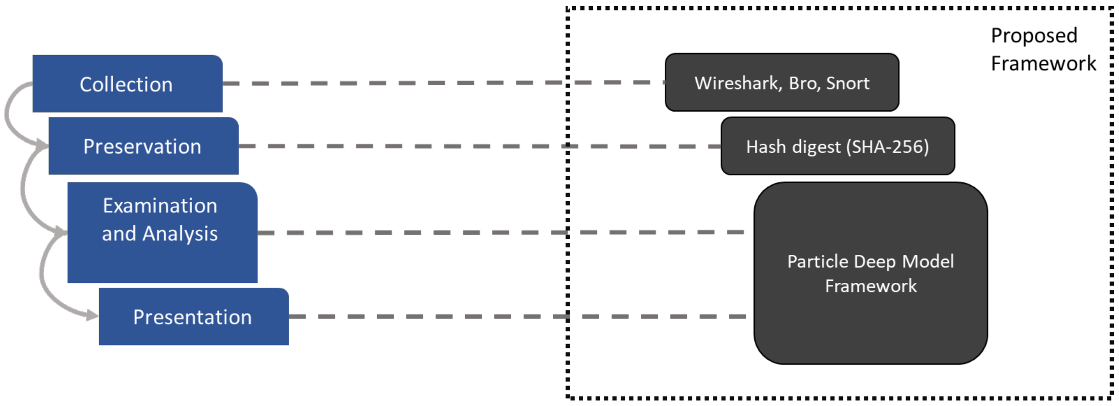


Fig. 5.2: Stages of investigation in network forensics including the proposed framework

an indication as to how well it performs when applied for this task. In other words, this research provides empirical data about PSO performance, when tasked with optimising deep MLP for application in the network forensic discipline.

5.5. Proposed Particle Deep Model for Network Forensics

The proposed particle deep model is an important addition to the field of network forensics to describe the stages of network forensics, namely collection, preservation, examination/analysis and presentation, as shown in Figure 5.2. The proposed model can be integrated to the investigation process of network forensics, by utilizing Deep Learning in the form of a neural network whose architecture is multi-layered, which in turn greatly improves its performance while maintaining a reasonable execution time.

Algorithm 5.2 presents the proposed particle deep model that integrates the PSO and MLP algorithms for improving the accuracy of attack detection and investigation and enhancing the computational process of the deep learning algorithm. The proposed model combines generative deep neural networks for the correct identification of malicious traffic, in a mixed environment comprised of both IoT and non-IoT traffic.

In order to determine the type of deep neural network to use in the PDF, the average performance of a vanilla deep MLP model [252] and Recurrent Neural Networks (RNN) with a different number of steps [52] were compared. The goal of these tasks is to build a model that is as accurate as possible while maintaining a low false alarm rate.

Algorithm 5.2 Particle deep model for hyperparameter estimation of deep learning

```
Data:
nn ← load_neural_network_structure() [b,e,lr] ← initialize_random_hyperparameters() hyperparameters ← [b,e,lr] PS ← construct_particle_swarm(n_particles,swarm_epochs) i ← 0 foreach
 $h_1 \in hyperparameters$ do
    while $PS.swarm\_epochs \neq 0$ do
        $h_1 \leftarrow PS.maximize(nn.AUC, h_1) using algorithm 1$
    end
    $nn.save\_opt\_hyperparam((h_1))$
end
$nn.train\_NN(training\_set())$
```

In Algorithm 5.2 we depict a Particle Deep Model (PDM) iteration. First, the neural network is loaded with its pre-selected layers and number of neurons. Initially, the three hyperparameters batch size, number of epochs and learning rate [b,e,lr] are randomly initialized. Next, a particle swarm comprised of a pre-selected number of particles (*n_particles*) and number of iterations (*swarm_epochs*) is generated. Then algorithm 5.1 is utilized to identify the value of the hyperparameter that is being optimised, that maximizes the AUC value of the neural network ($h_1 \leftarrow PS.maximize(nn.AUC, h_1)$). The process is repeated for every hyperparameter that is being optimised, the identified values of which are utilized to train the final neural network.

The Bot-IoT dataset [265] was utilized for training and testing of the deep models, with 80% of the dataset used for training while the remaining 20% for testing. On both the training and testing set, we performed min-max normalization, which resulted in the values being scaled data within the range of [0,1] to assert the neural networks models will not bias towards a particular class and ensure regularization of learning. We selected the type of deep learning model for our framework, and then we initially trained and validated it manually, through a trial-and-error process. After that, we employed the particle deep model, as explained in Algorithm ~5.2, to optimise the hyperparameters of the deep neural network. The reasoning behind this action is that there is no standard process for selecting the best values for hyperparameters, such as *the number of layers, the number of nodes for each layer, a learning rate*, during the pre-training phase of a neural network.

In the proposed particle deep model, the particles propagate through the search space of each of the three hyperparameters being optimised, one at a time. The logic behind individualizing the hyperparameter search was to make the search space smaller, since if the swarm was applied to all three hyperparameters at the same time, the search space would have been equal to *Batch_size_Size * Epochs_Size * Learning_rate_Size*, where *_Size* indicates all the possible values of the hyperparameter that can be estimated in their search space.

Hyperparameters values affect the training process of a neural network, and although there are many, we will focus on three, namely *epochs, batch size* and *learning rate*. The batch size determines the number of records (rows, if the data is in a structured form), which is parsed by the model before its weights are updated. The number of epochs indicates the times that the network will take the entire training dataset. The learning rate is a decimal, usually between (0,1), that is used to determine how much the weights are updated, with values close to 1 causing large updates that may be erratic and overshoot optima, while values close to 0 translates to very slow updates.

The logistic cost function was chosen, as it is suited for separating between attack and normal traffic, which is considered a binary classification problem [318]. Furthermore, due to class imbalance imposed by the nature of the attacks which have more records than normal traffic, weights were used to compensate for the imbalance. The logistic cost function is given in the following equation:

$$C = -\frac{1}{m}\sum_{i=1}^{m}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{5.5.1}$$

Where m is the number of instances that have been fed to the model, y_i is the expected class feature value, and $\hat{y}_i$ is the calculated class feature value of the i^{th} example.

The weights that were introduced for the two classes would be incorporated in the logistic loss equation like so: $w_1 y_i log(\hat{y}_i) + w_0(1 - y_i)log(1 - \hat{y}_i)$, with w_0 being the weight for the normal traffic and w_1 for the attack vector. Each particle trains a version of the deep MLP, with a different value for the trained hyperparameters, retaining their global best hyperparameters. The steps of training and validating data collection using the proposed particle deep model is discussed in Algorithm 5.3.

Because each particle needs to train an MLP to produce the AUC value, which is used in the search space to calculate speed and position for the particle, the PDF

Algorithm 5.3 Steps for training and testing the proposed particle deep model

```
Data:
$S = 0 :' batch', 1 :' epochs', 2 :' learning_rate'$  $Results = 'batch' : -1, 'epochs' : -1, 'learning_rate' : -1$  $NN = loadNN_structure()$ #Hyperparameters that aren't trained $n\_p = 6$ #number of particles $n\_e = 4$ #number of epochs $Results = randomInitialState()$ $task = ‘maximize\_AUC'$ for $k = 0; k <= 2; k++$ do
    #Runs once for each hyperparameter to optimise $particles = generateParticles(n\_p, n\_e)$ $bestHyper = runPSO(particles, task, S[k], NN)$
    $Results[S[k]] = bestHyper$
end
#trains a model with the identified hyperparameters $trainedNN = trainNN(NN, Results)$ $testNN(trainedNN)$
```

runtime is excessive. For each hyperparameter, the PSO execution time was as follows: for batch optimisation 4 hours, for the number of epoch optimisation 3 hours and for learning rate optimisation 4 hours, in total requiring 11 hours for optimisation. Additionally, training of the final deep MLP model required 7 minutes, while the prediction speed was 14,762 records/second. Regarding the time complexity of the proposed PDF, it is equal to $O(n_e * n_p * n_h) + O(mlp)$, where n_e is the number of epochs, n_p the number of particles that the PSO will use and n_h the number of hyperparameters to be optimised.

5.6. Architectural Design of Deploying proposed PDF in IoT Networks

In this section, we discuss an architectural design that illustrates the deployment of the proposed particle deep framework in IoT networks of smart homes, as an example of current smart systems, as shown in Figure 5.3. A typical IoT system architecture can be organized into three groups, the *IoT layer, network layer and the cloud layer* [270].

Our proposed framework could be easily deployed at the network layer, as its actions focus on tracing and discriminating between normal and attack vectors. Network traffic is usually encrypted on the Internet and real-world production networks, which hinders the analysis of payload data, while privacy laws may

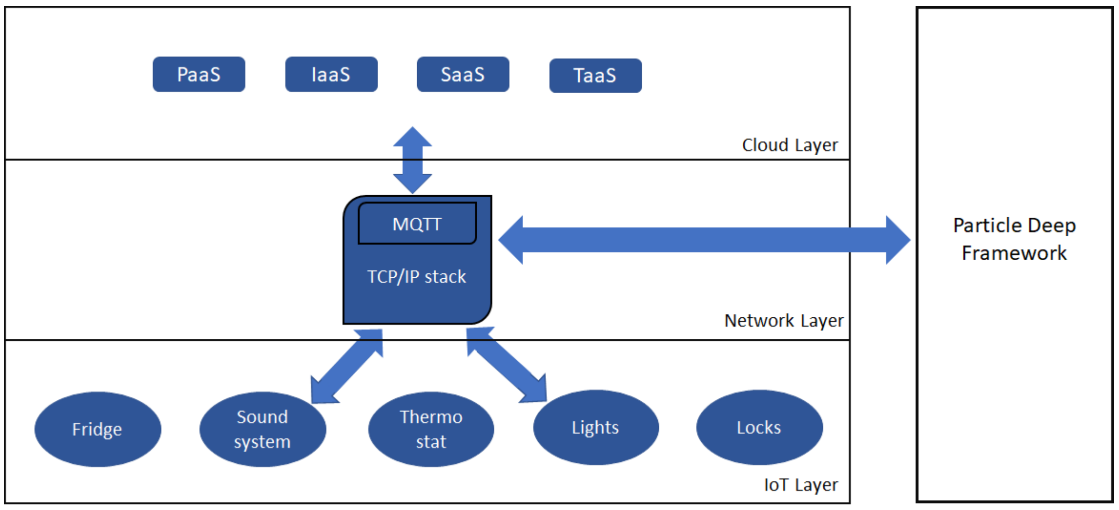


Fig. 5.3: Architectural design of deploying the proposed network forensic framework in IoT networks

also cause problems. As such, network flow analysis is used in this research to train and validate the proposed framework to avoid Law enforcement and privacy restrictions.

In the IoT layer, smart devices operate and interact with one another, via a local communication protocol such as ZigBee, Bluetooth or WiFi. Collected data is transmitted and commands that are issued through the network layer, which typically involves a coordinator and a sensor bridge device on the IoT layer side. A protocol that is often used by IoT systems is Message Queue Telemetry Transport (MQTT). MQTT is an application layer protocol that sits on top of the TCP/IP stack and enables high-speed and low bandwidth network communication thus lowering power requirements for the devices involved [319]. It is worth mentioning here that MQTT is stacked under TCP, so attacks such as DoS, DDoS, scanning could be types of MQTT attacks.

The MQTT protocol is utilized, in order to publish and subscribe to network data that reach the Cloud layer, where users can access the collected data, or manage their IoT services. The Cloud layer is organized into four categories depending on the type of service they provide, which are Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS), Software-as-a-Service (SaaS) and Thing-as-a-Service (TaaS) [320].

SaaS provides ready cloud-based software over the Internet, that is maintained by a third party and can be readily accessed by users. PaaS provides a platform, which comes with the tools necessary, including storage and cloud resources, for software development and maintenance. IaaS provides direct access to remote resources, either in virtual or in physical form, that the users operate and manage. TaaS is a newer concept, where network and data storage and analysis services are provided, in a way that makes them easy to integrate with deployed IoT systems. In this, TaaS resembles PaaS, as both provide configured platforms for development and analysis.

Essentially, a smart home is designed to offer comfort and efficiency through automation. As with IoT devices themselves, smart home deployments come in many shapes and forms. As such, they may be comprised of smart locks and music systems, which can be remotely activated and managed. For example, a realistic smart home deployment that was employed by Koroniotis et a. [265] to develop the Bot-IoT dataset, which has been used for this research, consisted of five IoT devices. Specifically, the IoT devices include a smart fridge, smart air-conditioning system, smart thermostat, smart lights and smart garage door. The proposed framework could be used to investigate security events involving Botnet activity and their origin in smart homes and other smart systems.

5.7. Experimental results and discussions

5.7.1. Datasets used and evaluation criteria

The Bot-IoT dataset [265] incorporates normal and attack traffic, including IoT traffic that was generated from Node-Red with 72.000.000 records and at 16.7 GB for the finalized dataset and combined pcap files at 69.3 GB. The UNSW-NB15 dataset [57] was generated by using the IXIA PerfectStorm tool, generating a number of diverse attacks, with the pcap files totalling at 100GB. We used these two datasets in this research, as they are both relatively new and both represent realistic normal traffic and attack scenarios. For training and testing, the Bot-IoT dataset was split in 80% and 20% respectively, resulting in 2.934.817 training and 733.705 testing records.

To evaluate and compare the trained models' performance, the following metrics are used:

- **Accuracy:** The fraction of correctly classified records from the total number of records.

$$(TP + TN)/(TP + TN + FP + FN) \tag{5.7.1}$$

- **Precision:** The fraction of records correctly classified as "Positive" from the records predicted as "Positive".

$$TP/(TP + FP) \tag{5.7.2}$$

- **Recall:** The fraction of records correctly classified as "Positive" from the total number of records that were "Positive".

$$TP/(TP + FN) \tag{5.7.3}$$

- **FPR:** The fraction of records falsely classified as "Positive" from the number of records that were "Negative".

$$FP/(FP + TN) \tag{5.7.4}$$

  FNR: The fraction of records falsely classified as "Negative" from the number of records that were "Positive".

$$FN/(FN + TP) \tag{5.7.5}$$

- **F-measure:** The harmonic mean of Precision and Recall.

$$2TP/(2TP + FP + FN) \tag{5.7.6}$$

The proposed particle deep framework was developed on a laptop outfitted with 16GB RAM, Intel Core i7-6700HQ CPU @2.6GHz and an NVIDIA GeForce GTX

970M. Python code was used to build and train the DNN model, as-well-as identify the hyperparameters through PSO. Specifically, the following python packages were used: Numpy and Pandas for matrix manipulation and data pre-processing, Keras which provided a high-level interface with the TensorFlow backend package and Optunity for the PSO process [321].

5.7.2. Results and Discussions

The experimentation results of evaluating the proposed Deep Particle Framework (DPF) using the evaluation metrics are explained in this subsection. In order to determine the neural network to use, a number of different architectures were tested, with their results presented in Table 5.1. The hyperparameters utilised are "epochs": "2", "batch_size": "512", "learning_rate": "0.001", with the activation function of hidden layers being "relu" and for the output layer "sigmoid". These hyper-parameters are by no means ideal for training but were arbitrarily chosen to help while comparing the effectiveness of the models. As such, we fed the output of the model to a single layer, single neuron Perceptron, with activation function "sigmoid", to obtain meaningful results.

Neural networks have been used to great success in a number of diverse fields, including cybersecurity and network forensics, proving their robustness and flexibility [322, 323]. Among the various neural network versions, the Multi-Layer Perceptron (MLP) is considered to be simple but powerful as it is capable of modelling data that is non-linearly separable. Similarly, two neural networks that incorporate the concept of memory in their inner workings are the Recurrent Neural Network (RNN) and the Long-Short Term Memory RNN (LSTM-RNN), which have been applied in a significant portion of the work related to cybersecurity [324, 325].

Recurrent Neural Networks, for instance, have been employed in various Intrusion Detection System deployments [157, 323]. Long-Short Term Memory RNNs improve on the RNNs by having the added ability to regulate when the network's memory should be updated, thus displaying better performance when classifying data in sequence. As this work seeks to identify the best neural network to use for the task of detecting Botnet activities in network captures where IoT devices were active, an empirical method was employed, where unoptimised neural networks were trained and tested on the same data. Then, by comparing their performance, the neural network best suited for the task of identifying botnet traces and differentiating them from normal traffic was identified. As such, the results of this empirical study are presented in Table 5.3.

An attempt was made to train an RNN with timesteps equal to Records, grouped by attack category, but handling this models output proved to be difficult. Additionally, the third, and deepest model could not be trained for timesteps=#records, as the memory requirements exceeded the existing RAM. As such, we grouped the Records in a window of 10 and trained the models (timesteps=10_Records).

Choosing timesteps=10 for the Bot-IoT dataset yielded the worst results. Even though it mostly displayed high values of accuracy (approximately 99.9% for most models), that value was misleading, as after considering the other metrics (specificity, recall, precision and NPV) and viewing the confusion matrices, it became evident that the model could not identify the negative class (normal traffic).

It should be noted here, that in our experiments the models that were trained appear to achieve good performance, with high accuracy, precision and recall, with small values of FNR, but also indicate high values of FPR. This can be explained by the fact that the models that were tested were not optimised, and that the Bot-IoT dataset has an imbalance, with more records representing attack scenarios that normal network traffic. Thus, by misclassifying normal records as attack instances, the classifiers minimised the error value used during training. To avoid this issue while training the final three deep MLP models, which are compared in Table ~5.4, weights were introduced to the logistic cost function.

In order to identify the best model to be used for the Network Forensic framework, Table ~ 5.3was constructed, which arranges the tested models in order, based on their average values, for the aforementioned five metrics (Accuracy, Precision, Recall, Specificity and Negative Predictive Value), as given in Table ~ 5.2.

Table 5.1: Produced Metrics, organized by model structure and type.

Models\metrics			Accuracy	Precision	Recall	FPR	FNR	F-measure
Input: 13 (1*) Hidden: 10, 4 Output: 1	MLP	-	0.999	0.999	0.999	0.728	$9.913*10^{-6}$	0.999
	RNN	Timestep=1	0.999	0.999	0.999	0.796	$8.674*10^{-6}$	0.999
		(*)Timestep= Features	0.999	0.999	0.999	0.699	$4.956*10^{-6}$	0.999
		Timestep=10_Records	0.999	0.999	1	1	0	0.999
Input: 13 (1*) Hidden: 10, 8, 9, 6, 4 Output: 1	MLP	-	0.999	0.999	0.999	0.543	$3.717*10^{-6}$	0.999
	RNN	Timestep=1	0.999	0.999	0.999	0.388	$6.196*10^{-6}$	0.999
		(*)Timestep= Features	0.999	0.999	1	1	0	0.999
		Timestep=10_Records						
Input: 13 (1*) Hidden: 20, 18, 10, 8, 4 Output: 1	MLP	-	0.999	0.999	0.999	0.223	$1.115*10^{-5}$	0.999
	RNN	Timestep=1	0.999	0.999	0.999	0.058	$3.841*10^{-5}$	0.999
		(*)Timestep= Features	0.999	0.999	0.999	0.174	$1.858*10^{-5}$	0.999
		Timestep=10_Records	0.999	0.999	0.999	0.917	$4.956*10^{-6}$	0.999
Input: 13 (1*) Hidden: 40, 26, 18, 14, 10, 8, 6, 4, 2 Output: 1	MLP	-	0.999	0.999	0.999	0.359	$1.115*10^{-5}$	0.999
	RNN	Timestep=1	0.999	0.999	0.999	0.126	$1.858*10^{-5}$	0.999
		(*)Timestep= Features	0.999	0.999	1	1	0	0.999
		Timestep=10_Records	0.999	0.999	1	1	0	0.999
Input: 13 (1*) Hidden: 8, 2 Output: 1	MLP	-	0.999	0.999	0.999	0.233	$1.734*10^{-5}$	0.999
	RNN	Timestep=1	0.999	0.999	0.999	0.64	$1.487*10^{-5}$	0.999
		(*)Timestep= Features	0.999	0.999	0.999	0.203	$1.982*10^{-5}$	0.999
		Timestep=10_Records	0.999	0.999	0.999	0.603	$1.115*10^{-5}$	0.999

Table 5.2: Average metrics.

Models \Metrics	Avg. Acc	Avg. Precision	Avg. Recall	Avg. FPR	Avg. FNR	Avg. F-measure
MLP	0.999	0.999	0.999	0.417	$1.065*10^{-5}$	0.999
RNN_Timestep=1	0.999	0.999	0.999	0.401	$1.734*10^{-5}$	0.999
RNN_Timestep=Features	0.999	0.999	0.999	0.615	$8.674*10^{-6}$	0.999
RNN_Timestep=10	0.999	0.999	0.999	0.903	$3.395*10^{-6}$	0.999

Table 5.3: Relative Position of models, with regards to the metrics and their performance.

	1^{st}	2^{nd}	3^{rd}	4^{th}
Avg. Acc	MLP, RNN_Timestep=1	RNN_Timestep=Features	RNN_Timestep=10	-
Avg. Precision	MLP, RNN_Timestep=1	RNN_Timestep=Features	RNN_Timestep=10	-
Avg. Recall	RNN_Timestep=10, RNN_Timestep=Features	MLP, RNN_Timestep=1	-	-
Avg. FPR	RNN_Timestep=1	MLP	RNN_Timestep=Features	RNN_Timestep=10
Avg. FNR	RNN_Timestep=Features	RNN_Timestep=10	RNN_Timestep=1	MLP
Avg. F-measure	MLP	RNN_Timestep=1	RNN_Timestep=Features	RNN_Timestep=10

As can be seen, the best performance on average was achieved by the RNN with 1 timestep model. Second, best was MLP and RNN with timeteps= features, while the worst performance was displayed when an RNN was trained with timesteps=10 (data appropriately re-shaped). It can be argued that an RNN with timestep=1 is equivalent to an MLP, with some added weight. As such, and based on the performances displayed here, the model best suited for the task of creating a network forensics framework appears to be the MLP model type.

Table 5.4: Neural Networks that were trained.

	(i) Unoptimised NN	(ii) Optimised NN with compressed input	(iii) Optimised NN with 13-features input
Neurons per layer	13, 20, 40, 60, 80, 40, 10, 1	1, 20, 40, 60, 80, 40, 10, 1	13, 20, 40, 60, 80, 40, 10, 1
Epochs	2	12	12
Batch size	350	3064	732
Learning rate	0.2	0.0015	0.0015
Accuracy	0.999	0.947	0.999
Precision	0.999	0.999	1
Recall	0.999	0.947	0.999
FPR	0.884	0.081	0
FNR	$9.269*10^{-5}$	0.052	$9.541*10^{-5}$
F-measure	0.999	0.973	0.999

A method for compressing the data was employed to investigate its effects on the performance metrics of the models. The process involves first passing each feature value through the Probability Density Function of the Normal Distribution, then multiplying the records (row-wise) with weights which were obtained by averaging the correlation coefficient matrix of the features and then adding the values, combining them into a single min-maxed normalized feature.

In order to tackle the class imbalance present in the Bot-IoT dataset between normal and attack traffic, we applied weights. Namely, the weights used were "1" for attack instances and "4500" for the normal instances. Following that, by employing the PDF, we obtained the following results for a deep neural network with the layers and neurons depicted in Table 5.4. Additionally, we explicitly set random seed values for both the PSO process and the neural network model, to ensure reproducibility. For the initialization of weights, we used *glorot uniform*.

In table 5.4, we present three trained neural networks. Starting with (i), this is an initial, unoptimised neural network, for which hyperparameters were arbitrarily chosen. Although it achieved high accuracy, the model performed poorly, as it achieved an FPR of 0.8846 (88.46%). The model's accuracy is high, due to an imbalance between normal and attack traffic in the Bot-IoT dataset. To compensate for this, during training, we applied weights as discussed above.

Next, (ii) is an MLP where the initial 13 features were combined into one. We pursued this option, to reduce the already considerable training time. The compression process was previously described. It should be noted, that the accuracy

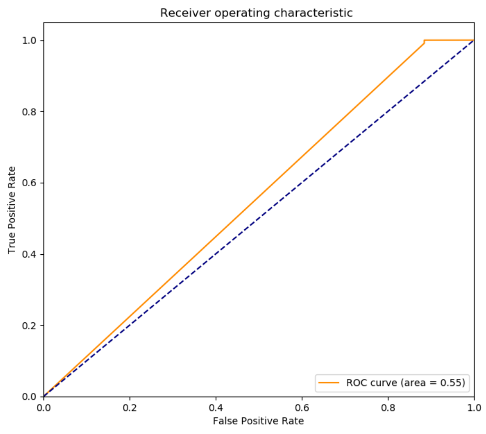


Fig. 5.4: (i) Unoptimised NN

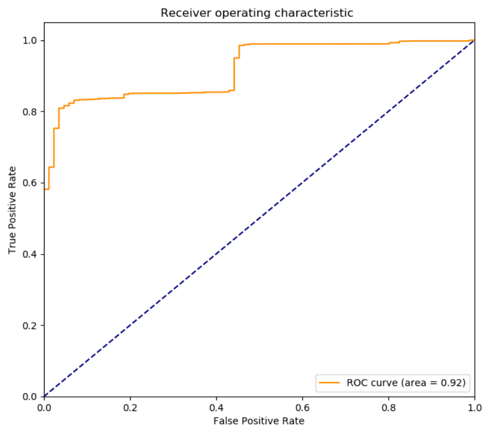


Fig. 5.5: (ii) Optimised NN with compressed input

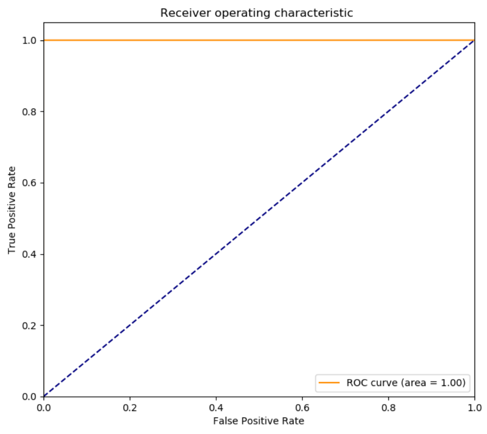


Fig. 5.6: (iii) Optimised NN with 13-features input

of close to 95% was achieved, although FPR and FNR were not decreased further than 8% and 5% respectively. Finally, (iii) represents the optimised MLP with the original 13 features. Compared to compressed input NN (ii), the only difference in the trained hyperparameter values was the batch size, which for the 13-feature input NN (iii) was a smaller 732. This network displayed the best performance out of all the networks we trained, while also reducing the FPR and FNR to close to 0.

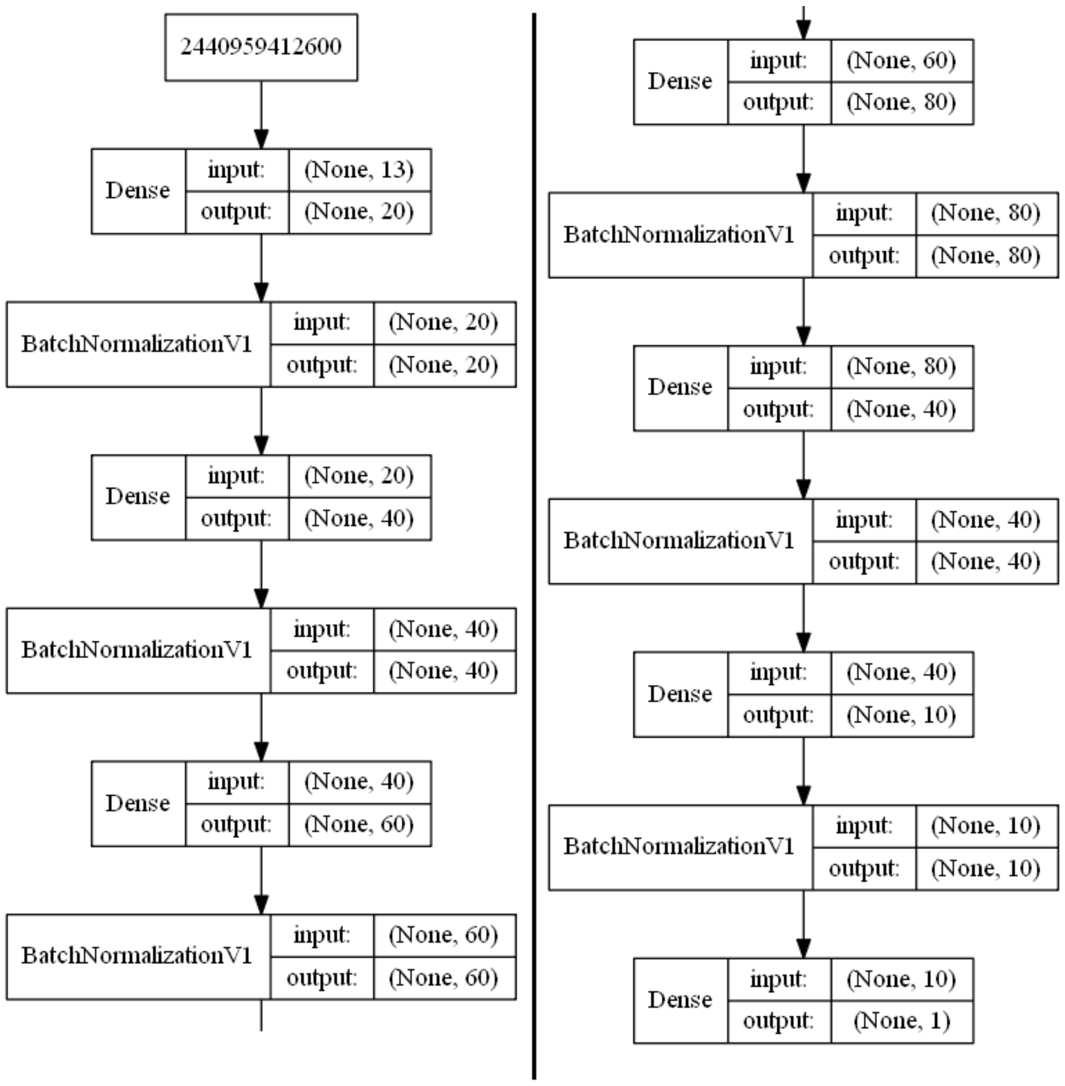

2440959412600
Dense input: (None, 13) output: (None, 20)
BatchNormalizationV1 input: (None, 20) output: (None, 20)
Dense input: (None, 20) output: (None, 40)
BatchNormalizationV1 input: (None, 40) output: (None, 40)
Dense input: (None, 40) output: (None, 60)
BatchNormalizationV1 input: (None, 60) output: (None, 60)
Dense input: (None, 60) output: (None, 80)
BatchNormalizationV1 input: (None, 80) output: (None, 80)
Dense input: (None, 80) output: (None, 40)
BatchNormalizationV1 input: (None, 40) output: (None, 40)
Dense input: (None, 40) output: (None, 10)
BatchNormalizationV1 input: (None, 10) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

Fig. 5.7: 13-Feature Model structure.

There are a few reasons why the 13-feature NN (iii) outperformed the compressed input NN(ii). To begin with, the single input model compressed the available information from 13 features, quite possibly causing significant information loss. Additionally, the 13-feature MLP had more connections, and thus weights between the input layer and the first hidden layer. Specifically, (iii) had 260 trainable weights, while the compressed input NN (ii) had 20, which can lead the former to learn more complex patterns in the data. Figure 5.7 depicts the 13-feature deep MLP (iii) model that was obtained by using the PDF.

From a network forensics point of view, the PDF is a useful tool. It can be used to detect attack traffic present in a packet capture file collected from a crime scene or a monitored network. By doing so, the origins of the attack can be established, assisting analysts with establishing a timeline of events, leading to the identification of the attack's target and sometimes the attacker's motive. Additionally, as it uses network flow information instead of performing deep packet inspection, privacy concerns are nullified. All of these concerns are important when introducing digital evidence in a court of law.

The PDM has some advantages, for example, it automates the process of defining hyperparameters, arguably a difficult process as there exists no explicit way of defining a "good" set of hyperparameters. Furthermore, the used deep architecture makes the model capable of capturing complex patterns in data, improving its expressiveness. On the other hand, one disadvantage of the PDM is that it is time-consuming. Each particle needs to train a new version of the network on the newly discovered hyperparameter as it traverses the search-space. One way to combat this and reduce time is by parallelizing each generation, so as to make all the particles run at the same time.

5.8. Attack scenarios that validate the PDF

The dataset that was used to train the deep MLP model through the proposed PDF, was the Bot-IoT dataset [265]. As such, the attack scenarios that can be detected by the finalized deep classification model are presented here.

- Probing attacks [199] - are malicious information-gathering activities, utilized by hackers in order to scan remote systems. Probing can be classified as either passive, where an attacker simply records traffic that is being exchanged by legitimate users and active, where an attacker exchanges traffic with the victim, gathering information about the latter's system through its responses. Depending on the purpose of the probe, these attacks are further split into Operating System and Service fingerprinting.
- Denial of Service [181] - are attacks that attempt to disrupt legitimate services, that are being accessed by users remotely. These attacks are split into two main subcategories, Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks, which are carried out by organized and compromised machines named Bots. As a secondary classification, DDoS and DoS attacks

can be volumetric and protocol-based [202]. Volumetric attacks focus on flooding their target with network traffic, while protocol-based attacks abuse weaknesses found in Internet protocols. The attack scenarios present in the Bot-IoT dataset include both DDoS and DoS with the following protocols used: TCP, UDP and HTTP.

- Information Theft [204] - is a category of attacks where the goal of the attacker is to extract sensitive data from the victim's machine. Based on the type of data that is targeted, these attacks can be split into two groups, data theft and keylogging. In the case of data theft, the attacker seeks to compromise the security of a remote machine and establish a reverse connection, through which data may be exfiltrated. On the other hand, in the case of keylogging, the attacker installs custom software on the remote machine, that records the keystrokes of the machine's user, effectively stealing any passwords that may be typed. The attacks in the dataset include both keylogging and data theft (exfiltration) scenarios.

5.9. Comparisons with other network forensics models-based Machine Learning

In this section, we compare the results obtained by the deep MLP that was trained by using the PDF, with results previously reported in other research studies, presented in Chapters 3 and 4. In order to evaluate the effectiveness of various machine learning models, [100] used the UNSW-NB15 dataset. The obtained results are compared to the new, optimised model. In addition, [265] trained some machine learning models to evaluate the Bot-IoT dataset, which was used in this research as well. These models include SVM, RNN and LSTM. As can be seen, the new optimised model improves on the performance of previous implementations.

As can be seen in tables 5.5, 5.6, the proposed deep optimised neural network (deep MLP), outperforms the other implementations that were trained both in the Bot-IoT and the UNSW-NB15 datasets respectively. The deep model achieved the highest accuracy and F-measure, while also the smallest FPR, FNR values, as the PSO can precisely identify the best hyper-parameters of the MLP and then the MLP can accurately discover cyber attack vectors and their attack families, as explained below.

Table 5.5: Comparison of DNN with previous classifiers.

	13-Feature DNN model	SVM	RNN	LSTM
Accuracy	0.999	0.883	0.997	0.997
Precision	1	1	0.999	0.999
Recall	0.999	0.883	0.997	0.997
FPR	0	0	0.733	0.687
FNR	$9.541*10^{-5}$	0.116	$2.5*10^{-3}$	$2.492*10^{-3}$
F-measure	0.999	0.938	0.998	0.998

Table 5.6: Comparison of Bot-IoT generated DNN, with classifiers built on the UNSW-NB15 dataset.

	13-Feature DNN model	ARM	Decision tree	Naïve Bayes	Perceptron
Accuracy	0.999	0.856	0.932	0.727	0.639
Precision	1	0.908	0.948	0.92	0.642
Recall	0.999	0.895	0.944	0.627	0.984
FPR	0	0.255	0.09	0.095	0.97
FNR	$9.541*10^{-5}$	0.104	0.055	0.372	0.015
F-measure	0.999	0.902	0.946	0.746	0.777

One of the main reasons behind the high FPR and FNR displayed by the other ML models that were compared to the optimised deep MLP is that the realistic cyber-attacks that are represented in the data are very close to the normal traffic. In other words, the DoS and DDoS volumetric attacks display characteristics similar to high-volume normal traffic, thus making discrimination between the two difficult. Furthermore, deep architectures such as the one used in this paper (deep MLP), that are comprised of multiple layers and neurons in each layer, render neural network capable of recognizing more complex patterns in the data.

The unoptimised Perceptron, Association Rule Mining (ARM) and Naïve Bayes classifiers produced highly erroneous results for various reasons. To begin with, the Perceptron lacked the necessary complexity, which is introduced through more layers and neurons, to model the patterns in the data. The ARM seeks to identify strong rules in the data, which may lead to some subtle patterns to be overlooked.

The Naïve Bayes, a probabilistic classifier, relies on the assumption of mutual independence of the features given the class, which often hinders classification. The Decision Tree's (DT) performance was the closest to the optimised deep MLP. One possibility for its performance could be that DTs classify data by creating perpendicular lines to the axis/dimensions (features), and the data was mostly linearly separable.

Hyperparameter selection has tremendous impact on a neural network's performance. As selecting hyperparameters is not strictly defined by rigid rules that can guarantee best results, the process was automated in the PDF, with each hyperparameter tuned separately one after another. By tuning each hyperparameter separately, the dimensions of the search space that the swarm needs to traverse are reduced. As such, the swarm can run for fewer iterations and gradually build on previous results.

5.10. Identification of Attack families and their statistics

In the Bot-IoT dataset, a number of botnet-related attacks are represented in the flows. These attacks form three groups, information gathering, information theft and Denial of Service (DoS) and Distributed DoS (DDoS). Information gathering attacks are activities which allow an attacker to identify the number, type and version of services of a remote machine such as service scanning and OS fingerprinting. Information theft attacks make use of previously exploited machines to steal sensitive information. From there, a bot can download documents present on the computer, or it can start recording keystrokes, thus stealing credentials, for example, keylogging and data theft attacks.

DoS attacks seek to render the remote services of a server unusable, and DDoS are multiple instances of DDoS to disrupt resources through protocols such as TCP and HTTP. Figure 5.8 shows the detection rate of the attack types included in the Bot-IoT datasets. The proposed particle deep framework is capable of distinguishing between normal and attack traffic with an accuracy of about 99.9%. Prediction time for 733,705 records was 49.7 seconds, thus having a speed of prediction at 14,762 records/second. As such the model would be capable of identifying the attacks present in the dataset.

In the UNSW-NB15 dataset, the network traffic that is represented includes both normal and a number of attack types. In total, nine attack types are represented in the dataset, which can be grouped into the following groups: Information gathering, Service disruption and Remote access. Analysis and Reconnaissance

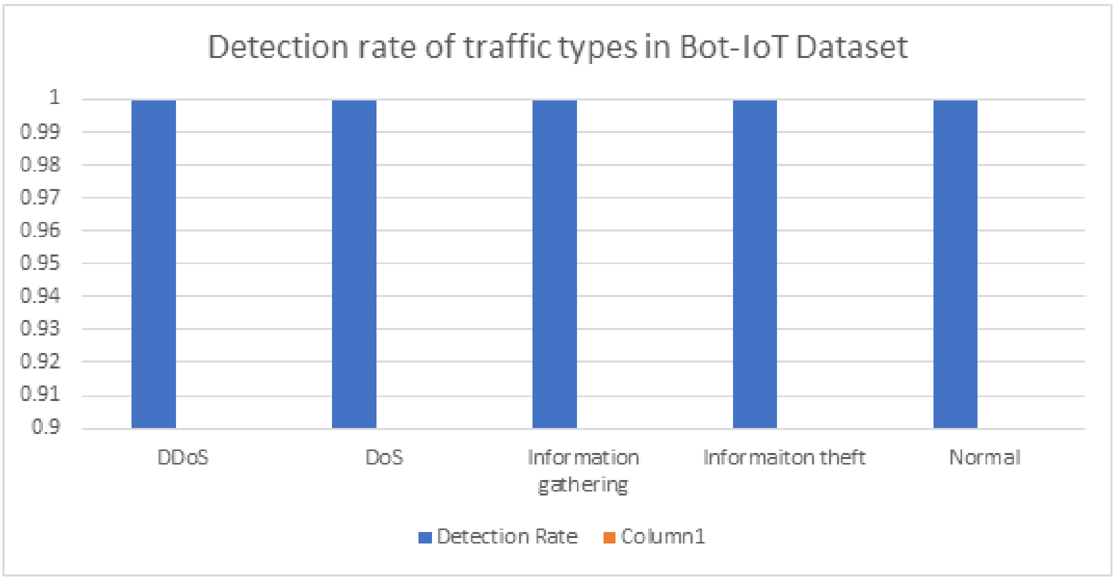


Fig. 5.8: Detection rate of attack types involved in the Bot-IoT dataset

traffic belongs to the Information gathering group, as they seek to extract information from the remote target, ranging from open ports to identifying programs and their version that has run in the target.

The Service disruption group included Fuzzers and DoS attacks. Fuzzers are tasked with causing a program to suspend its normal operation by feeding it randomly generated traffic. DoS attacks rely on flooding the remote host with legitimate traffic, either depleting resources through sheer volume of information, or by exploiting known protocol bugs. The rest of the attack traffic types (excluding Generic and Worm) are categorized under Remote access. These attacks include Backdoors where, after using an Exploit, a Shellcode is delivered as a payload to the remote target and a channel is established that allows stealthy access to the remote target. Finally, Generic is a cryptographic-based attack that tackles block-cyphers with a given block and key size and Worms, which are viruses that replicate itself and propagates to other computes in a network. The proposed framework can identify and trace all the attack types included in the UNSW-NB15 datasets with an approximate 99.2% as depicted in Figure 5.9.

5.11. Advantages and limitations of the PDF

The advantages of the proposed framework are the automation of hyperparameter selection, making manual optimisation obsolete. Additionally, the collection stage

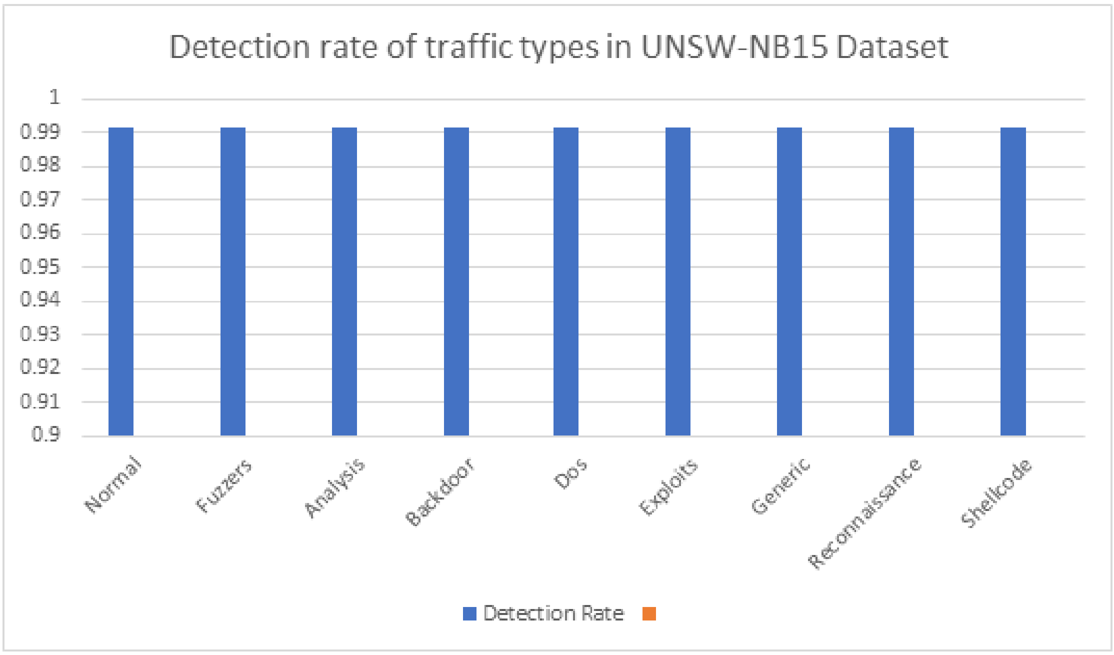


Fig. 5.9: Detection rate of attack types involved in the UNSW-NB15 dataset

utilizes software that is user-friendly and has been widely used in industry, thus it has credibility. Furthermore, the integrity of data is considered and maintained through functions, something that is very important, as forensic investigations can be nullified if it can not be ensured that the data has not been tampered. One disadvantage of the PDF would be the time of execution, as each move of a particle in each iteration requires time equal to the training time of the model we are optimising. As such, the PDF's execution time is dependant to the volume of training data. A possible remedy to this issue is the use of GPU and parallel computing for the training of different particles.

More relating to the underline data used to train the deep MLP, the simulated environment depicted several popular attacks, although it neglected to include any IoT-specific attacks, as the IoT devices were simulated through the use of Node-Red. The second limitation is related to the amount of time necessary to tune and train a NN through PDF. As each particle propagates through the search space, it effectively trains a version of the NN, with a different value for the hyperparameter that is being tuned. Thus, depending on the size of the data, and the architecture of the NN, time may be excessive. Finally, the PDF in its current form utilizes network flow data. Although flow data bypass some of the restrictions brought about by DPI, it relies on traffic statistics, ignoring information such as the payload which could help in the identification of attack traffic. Furthermore, overcoming spoofing attacks, where the source IP has been altered, is a challenge.

Nevertheless, the proposed framework displayed a very high accuracy of detection. One reason behind the PDF's performance is the choice of a deep neural network, which is suited to handling large quantities of data by default. Another reason is the utilization of PSO to identify the optimal hyperparameters that maximize the AUC of the model. Furthermore, compared to other frameworks [291, 293] that focused on malware detection, the PDF bases its functionality on processing network flow data, the statistics of which can be easily adapted for use by a neural network.

5.12. Differences of the Network Forensic PDF and traditional IDS

Network forensics encompasses a wide range of activities and tools that utilize network traces. Included in these tools are Network Intrusion Detection Systems (NIDS). NIDS are a type of construct, either hardware or software, that are designed to detect unauthorised access to a system [57]. Based on the mechanism used to distinguish between normal and attack traffic, NIDS are separated into signature and anomaly-based [57, 326]. The main differences between the two NIDS types are that signature-based rely on known attack patterns, while anomaly-based model normal behaviour and flag any deviations as attacks.

Although NIDSs and the PDF may share some similar underline mechanics, their functionality and use-cases differ significantly. To begin with, the PDF is a forensic tool that covers the major phases of a network forensic investigation, while a NIDS is primarily utilized as a security tool. The PDF can be used by an expert to collect, preserve and analyse network packets in accordance with the forensic principles. On the other hand, NIDS only monitor network traffic, raising alarms when they detect an abnormality, without collecting traffic or ensuring its integrity [327].

Furthermore, NIDS tend to have high false alarm rates (FAR) [326]. This phenomenon is explained by the way NIDS detect intrusions. Signature-based NIDS, maintain a database of signatures of known attacks, thus they are unable to detect unknown attacks. Anomaly-based NIDS model the legitimate traffic of an average user, flagging any deviations even if they are produced by a legitimate user. Additionally, some attacks that resemble legitimate traffic, such as dos attacks [328], may be confused as being legitimate by an anomaly-based NIDS. In contrast, the PDF employs an optimised deep learning model as its classification engine, capable of detecting attacks in network flows with very low FAR, and high accuracy.

Finally, it should be noted, that network forensic frameworks, like the PDF, are used to produce inferences on collected traffic. Through the analysis phase, connections between the identified attacks are discovered [270]. These connections may lead to the discovery of important information about the attacker, such as their capabilities, their target and their methods. In comparison, NIDS can not produce such results, although they can be combined with an Intrusion Prevention System, to better protect a network of devices from attacks [327]. Finally, the PDF incorporates in its procedure a feature extraction process along with a hyperparameter optimisation process based on PSO, which caused its very high accuracy and its near negligible error rates.

5.13. Conclusion

This chapter has introduced a new network forensics framework that is named Particle Deep Framework (PDF) for discovering cyber-attacks and tracing them in IoT networks. This is because Security incidents that target IoT networks have been on the rise, as industry and the public adopt this new technology. The stages of the network investigation process were detailed and the proposed PDF was described. A Particle Swarm Optimisation (PSO) has been used to adapt the best hyperparameters of Deep Learning. Then, a Multi-layer Perceptron (MLP) neural network algorithm has been trained and validated using the Bot-IoT and network datasets for evaluating the performance of the proposed PDF. The proposed framework achieves high performances in terms of detection accuracy and timely processing compared with other machine learning models. Specifically, a deep MLP model was trained and tested on the Bot-IoT dataset, achieving an accuracy of 0.999, FPR of 0 and FNR of $9.5*10^{-5}$ at a speed of 14,762 records per second. An architectural design that demonstrated the way of deploying the proposed framework in a network layer of IoT was described, using a smart home as an example of smart systems. The next chapter provides a summary of the work presented in this thesis, followed by future directions of research and concluding remarks.

Chapter 6

Conclusion

6.1. Introduction

This thesis makes a significant contribution in the fields of Network Forensics and Deep Learning, focusing specifically on Network Forensics for IoT environments. Most existing network forensic solutions for the IoT focused on the acquisition stage of an investigation, solving a significant problem, but ignoring the examination and analysis stages. However, differentiating between normal and attack traffic is an important step of the forensic procedure, as attack traffic is identified and is used to understand the target, means and motives of a cyber attacker. To that effect, training machine learning models is a viable solution for automating significant parts of the examination and analysis. As a result, in the work presented in this thesis, machine learning techniques have been investigated and incorporated into the presented network forensic framework.

However, developing forensic methods for IoT environments carry several implementation challenges which may hinder their effectiveness. First, the lack of standardization in the IoT means that each developer may implement IoT applications differently. As such, not only does this cause interoperability problems between products from different vendors, it also hinders the creation of a unified forensic solution that can be applied to any IoT device, regardless of the underline technologies. Second, the high speed with which the IoT generates network traffic, along with its capability to produce huge volumes of data, hinders network forensic investigations, as useful traffic is often either short-lived on the Internet, or buried in non-useful data. Furthermore, in order to evaluate the effectiveness of network forensic frameworks, especially for the examination and analysis stages, the use of a high-quality dataset is of utmost importance. Unfortunately, researchers face low availability of datasets to work with, with available datasets either being outdated, missing ground truth or not including IoT traffic.

The research study presented in this thesis provides significant contributions to the body of research, addressing the aforementioned challenges to a considerable degree, with some of the limitations providing future directions for research, as discussed in Sections 6.3 and 6.4 respectively. The main components of the proposed Particle Deep Framework, are a data source, pre-processing, hyperparameter optimization and a deep learning model.

Initially, in Chapter 3 existing network datasets, and their potential applicability to this research, were investigated. A review of the identified available network datasets revealed that they were not suitable for the needs of this research, as some were outdated, others lacked labels and yet others lacked IoT network traffic. As a result, the need to develop a new dataset which would then be used in this research became apparent. Initially, a suitable testbed was designed and constructed, comprised of virtual machines, which were chosen for their portability, inexpensiveness and deployment speed. The testbed was then utilized to generate a new dataset, called Bot-IoT, which combined current Botnet attack scenarios with IoT traffic. During the generation of this new dataset, new features were generated, and feature selection was performed through a combination of statistical methods, namely Correlation Coefficient and Joint Entropy. Analysis indicated high performance for machine learning and deep learning models trained on this new dataset, although, improvements could be made, something which was addressed in Chapter 5.

Secondly, in Chapter 4 the performance of both supervised and unsupervised machine learning models tasked with distinguishing between normal and attack traffic was investigated. For the purposes of this, three big data collections were investigated, namely NSL-KDD, UNSW-NB15 and the new Bot-IoT. The two other datasets were selected, due to their popularity, with NSL-KDD being one of the most broadly used datasets for evaluating network forensic solutions to date, while UNSW_NB15 is one of the newest network datasets available for use by the research community. Experimental results, obtained by training 5 supervised models, including a deep learning model, and 2 unsupervised models, support the applicability of machine learning for network forensics. Furthermore, by comparing the performance of the trained models, the quality of the new Bot-IoT dataset is established. Although the acquired results support the claim that machine learning is a viable solution for network forensic applications, further improvement is required, as it is of vital importance to minimize the false positive and false negative rates.

Finally, in Chapter 5, a novel Network Forensic Framework for investigating Botnet activities in the IoT was designed. The new framework, called Particle Deep Framework (PSO), is based on a Deep Neural Network (DNN). The framework covers various stages of the network forensic investigation process, including traffic collection, examination and analysis. To improve its performance, and because there exists no standard method for selecting optimal hyperparameters, which affect greatly the performance of machine learning models, a new method, based on Particle Swarm Optimisation (PSO) was applied. This optimization method automates the selection of hyperparameters, using performance metrics to guide the movement of particles in the search space. Evaluating the performance of non-optimised and optimised DNNs, indicate a significant improvement in performance, minimizing the false positive and negative rates, while maximizing the accuracy.

The rest of this chapter is organized as follows. In Section 6.2 the key contributions of this research are provided, while the limitations are presented in Section 6.3. Future directions for research are given in Section 6.4, with concluding remarks in Sections 6.5.

6.2. Contributions to Research

The key contributions of the work presented in this thesis are as follows.

- **Design and construction of a new virtual testbed for the generation of a new IoT network dataset, named Bot-IoT.** Several virtual machines were connected in the testbed, representing a realistic, real-world network and generating normal traffic along with Ostinato [62]. IoT devices were represented in the network through Node-Red [61]. Attacks were launched by using multiple tools found in the four Kali VMs, which played the role of the attackers, such as Metasploit [63] and Hping3 [64]. By using T-shark [65], pcap files of approximately 69 GB were captured, separated in 1 GB files to make them easier to process.

  Through Argus [191], and the use of scripts in the MySQL database, network flow information was extracted from the original pcaps, made up of 72,000,000 records and 43 features 1 binary class feature and 2 category/sub-category features which can be used for multiclass classification. From the original records, approximately 5% was extracted into csv files, separated in training and testing files. In total, there are 11 class values, a normal, and 10 attack types (OS scanning, service scanning, DDoS/DoS TCP/UDP/HTTP, Keylogging, Data theft). The dataset was evaluated through the use of machine learning, establishing its usability for network forensic applications.

- **Performing analysis on the new Bot-IoT dataset.** For the analysis of Bot-IoT, two high-quality datasets are chosen, the UNSW-NB15 and NSL-KDD. An initial analysis, to gauge the quality of the Bot-IoT, was performed on three models, SVM, RNN and LSTM. The models that were then chosen for the machine learning analysis, were five supervised models Association Rule Mining, Artificial Neural Network, Naïve Bayes, Decision Tree, Deep Belief Network and two unsupervised, namely K-Means and Expectation Maximization.

  Results from the initial analysis support the quality of the new Bot-IoT network dataset and its applicability to network forensic solutions through machine learning. Through the secondary machine learning analysis, it was shown that machine learning models can distinguish between attacks and normal traffic with high accuracy. These results support the use of machine learning for network forensic frameworks and work to expand the performance of these models, through deep learning, as done in contributions 3 and 4.

- **Identifying the optimal features for the deep learning model of the new Particle Deep Framework.** After generating the Bot-IoT dataset, comprised of the full 43 features, feature selection needed to be performed, in order to reduce its dimensionality and identify features that contribute the most to the classification process and thus improve the performance of machine learning models. This was achieved by employing statistical analysis and combining both the Pearson Correlation Coefficient (PCC), a measure of linear similarity and the Shannon Joint Entropy (SJE), a measure of dissimilarity/uncertainty between features.

  Initially, a pare-wise matrix for both PCC and SJE was calculated. Then, their average scores were calculated for each feature. These averages were then compared, and the 10 least-similar features, that is, features with high average entropy and low average correlation scores were selected. This process was applied twice on the data, initially on the auto-generated network flow features, which were produced by Ostinato [62] and then on a combination of the previous 10-best features and the 14 new features, thus arriving at the final 10-best feature version of the dataset.

- **Development of a new network forensic framework for investigating Botnet incidents in IoT environments, based on deep learning, called Particle Deep Framework (PDF).** The multiple stages of a network forensic investigation process were considered for the design of the PDF. Initially, acquisition is carried out through well-established software like Tcpdump [311] or Wireshark [310]. Then, through the use of cryptographic hashes, the collected information is preserved. Finally, for the stages of examination and analysis, the PDF utilizes deep learning, in the form of a Multi-Layer Perceptron (MLP).

  The many layers of the MLP, ensure that the neural network can better identify the underline patterns in the data, thus having improved performance, compared to shallower networks. An important part of training a neural network is determining its hyperparameters, like the number of epochs, batch size and learning rate. As there exists no standard methodology for selecting optimal hyperparameters that improve the networks performance, a method based on Particle Swarm Optimization (PSO) was employed. In this process, the particles iteratively traverse the search space for each hyperparameter, seeking to maximize the Area Under Curve (AUC) obtained from the Receiver Operating Characteristic (ROC) curve. Validating the process, yielded significant accuracy, precision and recall, while maintaining low false positive and false negative rates, with prediction speed at 14,762 records/second, thus producing results in a reasonable time.

6.3. Limitations

In this section, the limitations of the main components of the PDF are explained.

To begin with, the first limitation has to do with attack representation in the Bot-IoT dataset, specifically hacking IoT. In the virtual environment used to generate the Bot-IoT, a smart home IoT system was simulated, through Node-Red [61], with the smart IoT devices generating traffic to and from two MQTT brokers. However, because of the simulated environment set-up, attacks involving hacking and overcoming an IoT device's security were impossible to generate. To overcome this limitation, the testbed could be extended to include either virtual copies of IoT devices or physical devices.

The second limitation is the hyperparameter tuning time. Depending on the input data (training data), training a deep model can be time-consuming. This training time affects the optimization time, as presented in this thesis. The process of identifying optimal hyperparameters, involves particles training an MLP model, using their current position in the search space, thus obtaining the AUC value which the swarm attempts to maximize. Thus, in each epoch, each particle trains a version of the deep model, which is very time-consuming. One possible solution to this issue would be to employ parallelization techniques through threads and process for the particles in each epoch.

Finally, the PDF utilized network flow data to identify Botnet activities. Using flows, the PDF is capable of analysing traffic even if it is encrypted, by relying on the underline statistics of the communication. However, for the identification of source and destination of attacks, the PDF relies on the corresponding *source IP* and *destination IP* found in network flow records. In some cases, attackers may mask their IP addresses, altering it to a fake one through a process known as IP spoofing. In addition, some attacks utilize the IP spoofing technique as the basis for their attacks, for example in amplification attacks [329, 330]. A solution to this issue has been sought on a cybersecurity level through various schemes such as monitoring the temporal characteristics of traffic [331]. Nevertheless, this remains an open problem for research.

6.4. Future directions

In this section, we discuss research directions for future work based on existing challenges (of investigating botnets using forensics mechanism) that were not covered by this research. Having reviewed some of the work conducted in the discipline of network forensics, as described in Chapter 2, future directions of research, both generally in forensics for IoT and specifically related to the research presented in this thesis are explained in the following subsection, followed by open questions.

6.4.1. Issues to be resolved

- **Cloud storage of information-** Locating the evidence in an IoT Botnet-related security incident can also be challenging. In most implementations of the IoT, low-power physical devices are used as actuators, with local hubs and network nodes employed to gather and transport the collected information to a central Cloud Service provider. Through these Cloud Service providers, IoT services become available to users [1]. With this scheme in mind and knowing that actuators (the intelligent "things") are equipped with a limited amount of memory and power, data is quickly gathered and transferred to the Cloud, freeing up space in the actuators for further tasks to take place. As such, evidence will most probably be found in the Cloud, which introduces a new family of challenges to forensic investigations, among which jurisdiction limitations and conflicting laws are two prominent examples.

- **Honeypot development-** With some work already done in the field of building convincing Honeypots specifically targeting IoT-related adversaries [136, 137, 138], it is expected that further advances will be made. Some ways of enhancing Honeypot implementations might include, making them more resilient against anti-forensics mechanisms, increasing the number of supported protocols thus increasing the range of mimicked IoT devices and handling the massive quantities of incoming traffic which could be generated by an IoT Botnet.

- **Dealing with Variety, Veracity and Value of IoT data-** Although the field of network forensics as applied to the Internet of Things is still in development, by reviewing research done on this conjunction of fields (network forensics of IoT-botnets) it was observed that not much emphasis was given on dealing with the following problematic issues:

  - **Variety-** Data produced by the IoT may exist in either structured (database tables), semi-structured (XML, JSON) or unstructured (audiovisual files) form, depending on the type of device in question [163, 164]. At the same time, data produced by a single IoT system may be thematically heterogeneous. For example, in an automated home, a thermostat and some motion sensors can be connected to each other, so that when the motion sensors detect motion, the thermostat sets the room temperature to a preset value. Thus, with the possibility that in a single IoT system there can exist any combination of heterogeneous devices (Web cameras, digital locks, routers, thermostats), produced data, and thus also evidence will vary in format, making it necessary for a process capable of scanning diverse data and locating traces a necessity.

  - **Value-** With IoT introduced in several sectors of everyday life, such as home automation, the health domain and more, concerns about privacy arise [163, 164], as sensitive information is recorded, exchanged, maintained and stored by the IoT devices and their service providers. As such, forensic investigations need to be conducted with a level of transparency and steps must be taken to ensure that private data are not exploited.
  - **Veracity-** deployed in a dynamic world, it is easy for environmental conditions to change inexplicably, causing the validity of recordings from finely calibrated IoT sensors to be faulty and producing inaccurate, low quality or noisy data [163]. As such, identifying such problems with the collected data is a challenge, as "contaminated" data could lead to false results in an investigation.

These characteristics of the IoT make applying network forensic techniques established in conventional IT systems inapplicable, thus directing future research towards dealing with these challenges [163]. Future directions of research specifically related to the work presented in this thesis are as follows.

- In this research, PSO was utilised to select hyperparameters for a deep MLP model that is a part of the PDF. In the future, this work could be expanded, by applying the PSO hyperparameters selection method to other generative deep neural networks. By doing so, the capabilities of multiple deep models can be optimised in a similar fashion, allowing for their performance to be compared and the best model selected for network forensic applications

- The network forensic framework PDF was designed and deployed by using data simulating a smart home environment. As such, research can be performed to provide more insights for deploying the PDF in other real-world IoT networks. For example, the research could focus on applying the PDF in smart health networks, where cyber-attacks can have dire consequences.

- In the Bot-IoT, various up-to-date cyber-attacks are represented in the network traces. However, due to the virtualized nature of the testbed, some attacks could not be simulated. As such, future research may focus on either altering or re-designing the testbed, to incorporate physical IoT devices. By doing so, attacks that focus on exploiting hardware limitations, or software bugs present in the firmware could be generated, leading to an expanded dataset.

6.4.2. Open questions

In this subsection, open questions that may be used for future research are presented, based on the issues discussed in the previous subsection.

- How can the optimization process presented in the PDF be improved?
  - Are there better choices for the objective function, than using the AUC?
  - How can the optimization process be sped up?
- How can the PDF be utilised in other smart deployments?
- Can the Bot-IoT dataset be expanded to incorporate a more diverse range of cyber-attacks?
  - How can the proposed testbed be modified or updated, to incorporate physical IoT devices?
  - What are the attacks that specifically target IoT devices that can be generated?
- Can the PDF's functionality be extended to detect and combat IP spoofing?

6.5. Final Remarks

With the ever-increasing number of IoT devices connected to the Internet, and considering their inherent security limitations and vulnerabilities, as-well-as their always-on mode of operation, it is evident that they will continue to be targeted for attackers and botnets. As such, using forensics to investigate security incidents, where the IoT has been targeted or has taken part in the attack itself, is of utmost importance. However, due to the lack of standardization and common specifications, designing a unified forensic solution is very difficult. Thus, one viable strategy is to utilize network forensics, as most IoT deployments make use of the Internet to transfer data from the sensors/actuators to the Cloud backend.

This thesis makes significant contributions to the field of Network Forensics for investigating Botnets activities in IoT settings, by analysing captured network traffic through 1) generating the Bot-IoT dataset, a new dataset with up-to-date network attack traces and normal instances that incorporates IoT generated traffic; 2) analysing the Bot-IoT dataset, against two well known and widely used dataset, to establish its quality and test the applicability of machine learning models for the identification of network security events; 3) proposing a new network forensic framework, called PDF, that incorporates deep learning for the detection of botnet activities and Particle Swarm Optimization for the identification of optimal hyperparameters, in order to enhance the performance of the deep model.

In short, the proposed framework can be used to effectively detect botnet activities in network traces obtained from a network with IoT deployments. Through the PSO method, optimal hyperparameters were chosen, improving the performance of the DNN (MLP) network, effectively reaching an accuracy of close to 1, and false-positive/negative rates very close to 0. Furthermore, due to the use of the deep model, the PDF is capable of rapidly producing results, at a speed of over 14,000 predictions/second. Thus, the steps described by the proposed forensic framework, along with the empirical results obtained during testing, support the claim that the PDF can be used in real-world scenarios.

Bibliography

[1] Md Mahmud Hossain, Maziar Fotouhi, and Ragib Hasan. Towards an analysis of security issues, challenges, and open problems in the internet of things. In *Services (SERVICES), 2015 IEEE World Congress on*, pages 21–28. IEEE, 2015.

[2] SearchSecurity. Get the best botnet protection with the right array of tools., 2018. Accessed: 5/02/2018.

[3] Somayya Madakam, R Ramaswamy, and Siddharth Tripathi. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(05):164, 2015.

[4] Postscape. Iot standards & protocols guide | 2018 comparisons on network, wireless comms, security, industrial., 2018. Accessed: 4/02/2018.

[5] Sam Cook. 60+ iot statistics and facts. *Comparitech*, 2019.

[6] IoTAA. Internet of things alliance australia internet of things security guideline v1.0, February 2017.

[7] Rob van der Meulen. Gartner says 8.4 billion connected 'things' will be in use in 2017, up 31 percent from 2016, Feb 2017. Copyright - Copyright Â© Targeted News Service. All Rights Reserved; Last updated - 2017-02-08.

[8] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The Internet Society (ISOC)*, pages 1–50, 2015.

[9] Symantec Corporation. Internet security threat report, volume 24. Technical Report 24, Symantec, 2019.

[10] Hewlett Packard. Internet of things research study–2015 report. *Online: http://www8. hp. com/h20195*, 2, 2015.

[11] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 4(6):1899–1909, 2017.

[12] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[13] Amanda Fakhreddine Martin McKeay, Jose Arteaga et al. akamai's state of the internet security q4 2016 report. *Akamai Technologies*, 5(4), 2016.

[14] Christopher Robberts and Joachim Toft. Finding vulnerabilities in iot devices: Ethical hacking of electronic locks, 2019.

[15] Mark Pollitt. Computer forensics: An approach to evidence in cyberspace. In *Proceedings of the National Information Systems Security Conference*, volume 2, pages 487–491, 1995.

[16] Anamika Joshi and DS Bhilare. Digital forensics: Emerging trends and analysis of counter-security environment. *International Journal*, 3(12), 2013.

[17] Christopher Meffert, Devon Clark, Ibrahim Baggili, and Frank Breitinger. Forensic state acquisition from internet of things (fsaiot): A general framework and practical approach for iot forensics through iot device state acquisition. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 56. ACM, 2017.

[18] Mahmud Hossain, Yasser Karim, and Ragib Hasan. Fif-iot: A forensic investigation framework for iot using a public digital ledger. In *2018 IEEE International Congress on Internet of Things (ICIOT)*, pages 33–40. IEEE, 2018.

[19] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac. Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles. *IEEE Communications Magazine*, 56(10):50–57, OCTOBER 2018.

[20] Md Mahmud Hossain, Ragib Hasan, and Shams Zawoad. Probe-iot: A public digital ledger based forensic investigation framework for iot. In *INFOCOM Workshops*, pages 1–2, 2018.

[21] Chad Perrin. The cia triad. *Dostopno na: http://www. techrepublic. com/blog/security/the-cia-triad/488*, 2008.

[22] Pedram Amini, Muhammad Amin Araghizadeh, and Reza Azmi. A survey on botnet: Classification, detection and defense. In *Electronics Symposium (IES), 2015 International*, pages 233–238. IEEE, 2015.

[23] A. Rajan, J. Jithish, and S. Sankaran. Sybil attack in iot: Modelling and defenses. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2323–2327, Sep. 2017.

[24] Kishore Angrishi. Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. *arXiv preprint arXiv:1702.03681*, 2017.

[25] Georgios Kambourakis, Constantinos Kolias, and Angelos Stavrou. The mirai botnet and the iot zombie armies. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, pages 267–272. IEEE, 2017.

[26] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. Iot goes nuclear: Creating a zigbee chain reaction. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 195–212. IEEE, 2017.

[27] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654. IEEE, 2016.

[28] Bako Ali and Ali Awad. Cyber and physical security vulnerability assessment for iot-based smart homes. *Sensors*, 18(3):817, 2018.

[29] Elad Shuster. Financial services attack economy. *Akamai Technologies*, 5(4), 2019.

[30] Christian Beek Raj Samani. Mcafee labs threats report. Technical report, McAfee Labs, December 2018.

[31] Cinthya Grajeda, Frank Breitinger, and Ibrahim Baggili. Availability of datasets for digital forensics–and what is missing. *Digital Investigation*, 22:S94–S105, 2017.

[32] Livadas Carl et al. Using machine learning technliques to identify botnet traffic. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on. IEEE*, 2006.

[33] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. *arXiv preprint arXiv:1804.04159*, 2018.

[34] 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.

[35] Pedram Amini, Reza Azmi, and Muhammad Amin Araghizadeh. Analysis of network traffic flows for centralized botnet detection. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 11(2):7–17, 2019.

[36] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Ibrahim Baggili. Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, 10(1):34–43, 2013.

[37] Victor R Kebande and Indrakshi Ray. A generic digital forensic investigation framework for internet of things (iot). In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*, pages 356–362. IEEE, 2016.

[38] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 66:214–235, 2016.

[39] B Hazarika and S Medhi. Survey on real time security mechanisms in network forensics. *International Journal of Computer Applications*, 151(2), 2016.

[40] Jia-Rong Sun, Mao-Lin Shih, and Min-Shiang Hwang. A survey of digital evidences forensic and cybercrime investigation procedure. *IJ Network Security*, 17(5):497–509, 2015.

[41] Amor Lazzez and Thabet Slimani. Forensics investigation of web application security attacks. *Int. J. Comput. Netw. Inf. Secur*, 7(3):10–17, 2015.

[42] Marwa Keshk, Elena Sitnikova, Nour Moustafa, Jiankun Hu, and Ibrahim Khalil. An integrated framework for privacy-preserving based anomaly detection for cyber-physical systems. *IEEE Transactions on Sustainable Computing*, 2019.

[43] Nitin Naik, Paul Jenkins, Roger Cooke, and Longzhi Yang. Honeypots that bite back: A fuzzy technique for identifying and inhibiting fingerprinting attacks on low interaction honeypots. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2018.

[44] Vaclav Oujezsky, Tomas Horvath, and Vladislav Skorpil. Botnet c&c traffic and flow lifespans using survival analysis. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, 6(1):38–44, 2017.

[45] Ronald Cheng and Gavin Watson. D2pi: Identifying malware through deep packet inspection with deep learning. 2018.

[46] Prasanta Gogoi, Monowar H Bhuyan, DK Bhattacharyya, and Jugal K Kalita. Packet and flow based network intrusion dataset. In *International Conference on Contemporary Computing*, pages 322–334. Springer, 2012.

[47] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[48] R Gentleman and VJ Carey. Unsupervised machine learning. In *Bioconductor case studies*, pages 137–157. Springer, 2008.

[49] M Steinbach, V Kumar, and P Tan. Cluster analysis: basic concepts and algorithms. *Introduction to data mining, 1st edn. Pearson Addison Wesley*, 2005.

[50] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[51] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[52] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[53] Allen Huang and Raymond Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

[54] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[55] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[56] Unibs, university of brescia dataset, 2009.

[57] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*, pages 1–6. IEEE, 2015.

[58] Kddcup99 dataset.

[59] Iman Sharafaldin, A Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of fourth international conference on information systems security and privacy, ICISSP*, 2018.

[60] 1998 darpa intrusion detection evaluation data set.

[61] Node-red tool.

[62] Ostinato.

[63] Metasploit framework.

[64] hping.

[65] Tshark network analysis tool.

[66] G Hall. Pearson's correlation coefficient. *other words*, 1(9), 2015.

[67] Annick Lesne. Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics. *Mathematical Structures in Computer Science*, 24(3), 2014.

[68] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[69] Amanda Fakhreddine Martin McKeay, Jose Arteaga et al. akamai's [state of the internet] / security q4 2016 report. *Akamai Technologies*, 3(4), 2016.

[70] Neamen Negash and Xiangdong Che. An overview of modern botnets. *Information Security Journal: A Global Perspective*, 24(4-6):127–132, 2015.

[71] Kevin Ashton et al. That 'internet of things' thing thing in the real world, things matter more than ideas. *RFID journal*, 22(7):97–114, 2009.

[72] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. The internet of things in an enterprise context. In *Future Internet Symposium*, pages 14–28. Springer, 2008.

[73] CLUSTER CERP-IOT. Internet of things, strategic research roadmap. *European Commission*, 2009.

[74] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[75] Fred Donovan. A brief history of the internet of things, 2014.

[76] P Suresh, J Vijay Daniel, V Parthasarathy, and RH Aswathy. A state of the art review on the internet of things (iot) history, technology and fields of deployment. In *Science Engineering and Management Research (ICSEMR), 2014 International Conference on*, pages 1–8. IEEE, 2014.

[77] Ismael Peña-López et al. Itu internet report 2005: the internet of things. *ITU Internet report 2005*, 2005.

[78] Thomas Truong, Anh Dinh, and Khan Wahid. An iot environmental data collection system for fungal detection in crop fields. In *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*, pages 1–4. IEEE, 2017.

[79] Emir Husni, Galuh Boy Hertantyo, Daniel Wahyu Wicaksono, Faisal Candrasyah Hasibuan, Andri Ulus Rahayu, and Muhamad Agus Triawan. Applied internet of things (iot): Car monitoring system using ibm bluemix. In *Intelligent Technology and Its Applications (ISITIA), 2016 International Seminar on*, pages 417–422. IEEE, 2016.

[80] Mobyen Uddin Ahmed, Mats Björkman, Aida Čaušević, Hossein Fotouhi, and Maria Lindén. An overview on the internet of things for health monitoring systems. In *International Internet of Things Summit*, pages 429–436. Springer, 2015.

[81] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[82] Sérgio SC Silva, Rodrigo MP Silva, Raquel CG Pinto, and Ronaldo M Salles. Botnets: A survey. *Computer Networks*, 57(2):378–403, 2013.

[83] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. In *Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on*, pages 116–123. IEEE, 2013.

[84] Jane McCallion. Dell, fbi and nca bring down botnet behind Â£20m cyber bank heist. *IT Pro*, Oct 14 2015. Copyright - Copyright Dennis Publishing Ltd. Oct 14, 2015; Last updated - 2015-10-14.

[85] M. E. Kabay. Kraken the botnet: The ethics of counter-hacking. *Network World (Online)*, Mar 2009. Copyright - Copyright 2009 Network World, Inc. All Rights Reserved; Last updated - 2016-03-14.

[86] Robert Bradley Gilbert. *Defending Against Malicious Software.* University of California, Santa Barbara, 2011.

[87] Somayeh Soltani, Seyed Amin Hosseini Seno, Maryam Nezhadkamali, and Rahmat Budiarto. A survey on real world botnets and detection mechanisms. *International Journal of Information and Network Security*, 3(2):116, 2014.

[88] Defence Intelligence. Mariposa botnet analysis. Technical report, Technical report, October, 2009.

[89] Cui Xiang, Fang Binxing, Yin Lihua, Liu Xiaoyi, and Zang Tianning. Andbot: towards advanced mobile botnets. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, pages 11–11. USENIX Association, 2011.

[90] Paolo Farina, Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Are mobile botnets a possible threat? the case of slowbot net. *Computers & Security*, 58:268–283, 2016.

[91] Ruchna Nigam. A timeline of mobile botnets. *Virus Bulletin, March*, 2015.

[92] Sheharbano Khattak, Naurin Rasheed Ramay, Kamran Riaz Khan, Affan A Syed, and Syed Ali Khayam. A taxonomy of botnet behavior, detection, and defense. *IEEE communications surveys & tutorials*, 16(2):898–924, 2014.

[93] Anchit Bijalwan, Meenakshi Thapaliyal, Emmanuel S Piili, and RC Joshi. Survey and research challenges of botnet forensics. *International Journal of Computer Applications*, 75(7), 2013.

[94] K Narasimha Mallikarjunan, K Muthupriya, and S Mercy Shalinie. A survey of distributed denial of service attack. In *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*, pages 1–6. Ieee, 2016.

[95] Mohd Azahari Mohd Yusof, Fakariah Hani Mohd Ali, and Mohamad Yusof Darus. Detection and defense algorithms of different types of ddos attacks. *International Journal of Engineering and Technology*, 9(5):410, 2017.

[96] Yunus Yusoff, Roslan Ismail, and Zainuddin Hassan. Common phases of computer forensics investigation models. *International Journal of Computer Science & Information Technology*, 3(3):17–31, 2011.

[97] Ravneet Kaur and Amandeep Kaur. Digital forensics. *International Journal of Computer Applications*, 50(5), 2012.

[98] Ray Hunt and Sherali Zeadally. Network forensics: an analysis of techniques, tools, and trends. *Computer*, 45(12):36–43, 2012.

[99] Nour Moustafa and Jill Slay. Rcnf: Real-time collaborative network forensic scheme for evidence analysis. *arXiv preprint arXiv:1711.02824*, 2017.

[100] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Jill Slay. Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques. In *International Conference on Mobile Networks and Management*, pages 30–44. Springer, 2017.

[101] Nour Moustafa, Jill Slay, and Gideon Creech. Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks. *IEEE Transactions on Big Data*, 2017.

[102] Marwa Keshk, Nour Moustafa, Elena Sitnikova, and Gideon Creech. Privacy preservation intrusion detection technique for scada systems. *arXiv preprint arXiv:1711.02828*, 2017.

[103] Murray Brand, Craig Valli, and Andrew Woodward. Malware forensics: Discovery of the intent of deception. *Journal of Digital Forensics, Security and Law*, 5(4):2, 2010.

[104] Aparna Verma, MS Rao, AK Gupta, W Jeberson, and Vrijendra Singh. A literature review on malware and its analysis. *International Journal of Current Research and Review*, 5(16):71, 2013.

[105] Gary Palmer. A road map for digital forensics research-report from the first digital forensics research workshop (dfrws). *Utica, New York*, 2001.

[106] Venansius Baryamureeba and Florence Tushabe. The enhanced digital investigation process model. In *Proceedings of the Fourth Digital Forensic Research Workshop*, pages 1–9, 2004.

[107] Sujata Mittal and Rajvir Singh. A support vector approach for formulating ids rules using honeypot data. *Advanced Journal of Computer Science and Engineering (AJCST) ISSN : 2393-8390 (O)*, 4:1 – 5, 06 2016.

[108] Van-Hau Pham and Marc Dacier. Honeypot trace forensics: The observation viewpoint matters. *Future Generation Computer Systems*, 27(5):539–546, 2011.

[109] Sanjeev Kumar, Paramdeep Singh, Rakesh Sehgal, and JS Bhatia. Distributed honeynet system using gen iii virtual honeynet. *International Journal of Computer Theory and Engineering*, 4(4):537, 2012.

[110] Abigail Paradise, Dvir Cohen, Asaf Shabtai, and Rami Puzis. Generation of automatic and realistic artificial profiles. *arXiv preprint arXiv:1807.00125*, 2018.

[111] Jiman Jeong, Syed Moeen Ali Naqvi, and MyungKeun Yoon. Accurate and communication-efficient detection of widespread events. *IEEE Access*, 2018.

[112] Abirami Sivaprasad, Neha Ghawalkar, Srushti Hodge, Maitri Sanghavi, and Vidhya Shinde. Machine learning based traffic classification using statistical analysis. *International Journal on Recent and Innovation Trends in Computing and Communication*, 6(3):187–191, 2018.

[113] Dinil Mon Divakaran, Kar Wai Fok, Ido Nevat, and Vrizlynn LL Thing. Evidence gathering for network security and forensics. *Digital Investigation*, 20:S56–S65, 2017.

[114] Elias Bou-Harb and Mark Scanlon. Behavioral service graphs: A formal data-driven approach for prompt investigation of enterprise and internet-wide infections. *Digital Investigation*, 20:S47–S55, 2017.

[115] Anchit Bijalwan, Mohammad Wazid, Emmanuel S Pilli, and Ramesh Chandra Joshi. Forensics of random-udp flooding attacks. *JNW*, 10(5):287–293, 2015.

[116] Jerome Francois, Shaonan Wang, Walter Bronzi, Radu State, and Thomas Engel. Botcloud: Detecting botnets using mapreduce. In *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*, pages 1–6. IEEE, 2011.

[117] Lakshya Mathur, Mayank Raheja, and Prachi Ahlawat. Botnet detection via mining of network traffic flow. *Procedia Computer Science*, 132:1668–1677, 2018.

[118] Rafał Kozik. Distributing extreme learning machines with apache spark for netflow-based malware activity detection. *Pattern Recognition Letters*, 101:14–20, 2018.

[119] Abdurrahman Pektaş and Tankut Acarman. Botnet detection based on network flow summary and deep learning. *International Journal of Network Management*, page e2039, 2018.

[120] Duc C Le, A Nur Zincir-Heywood, and Malcolm I Heywood. Unsupervised monitoring of network and service behaviour using self organizing maps. *Journal of Cyber Security and Mobility*, 8(1):15–52, 2019.

[121] Zhen Chen, Fuye Han, Junwei Cao, Xin Jiang, and Shuo Chen. Cloud computing-based forensic analysis for collaborative network security management system. *Tsinghua science and technology*, 18(1):40–50, 2013.

[122] Chunyong Yin, Hongyi Wang, Xiang Yin, Ruxia Sun, and Jin Wang. Improved deep packet inspection in data stream detection. *The Journal of Supercomputing*, pages 1–14, 2018.

[123] Martin Holkovič, Ondřej Ryšavỳ, and Jindřich Dudek. Automating network security analysis at packet-level by using rule-based engine. In *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*, page 14. ACM, 2019.

[124] Sakshi Bansal, Mir Qaiser, Shefali Khatri, and Anchit Bijalwan. Botnet forensics framework: Is your system a bot. In *Advances in Computing and Communication Engineering (ICACCE), 2015 Second International Conference on*, pages 535–540. IEEE, 2015.

[125] Fuye Han, Zhen Chen, HongFeng Xu, Haopei Wang, and Yong Liang. A collaborative botnets suppression system based on overlay network. *International Journal of Security and Networks*, 7(4):211–219, 2012.

[126] KS Karthika. Peer to peer botnet detection system. *International Conference on Information and Image Processing (ICIIP)*, 2014.

[127] Ying Zhu. Attack pattern discovery in forensic investigation of network attacks. *IEEE journal on selected areas in communications*, 29(7):1349–1357, 2011.

[128] Duhoe Kim, Yong-Hyun Kim, Dongil Shin, and Dongkyoo Shin. Fast attack detection system using log analysis and attack tree generation. *Cluster Computing*, 22(1):1827–1835, 2019.

[129] David Gugelmann, Fabian Gasser, Bernhard Ager, and Vincent Lenders. Hviz: Http (s) traffic aggregation and visualization for network forensics. *Digital Investigation*, 12:S1–S11, 2015.

[130] Cliff Joslyn, Sutanay Choudhury, David Haglin, Bill Howe, Bill Nickless, and Bryan Olsen. Massive scale cyber traffic analysis: a driver for graph database research. In *First International Workshop on Graph Data Management Experiences and Systems*, page 3. ACM, 2013.

[131] Bijalwan Anchit and Singh Harvinder. Investigation of udp bot flooding attack. *Indian Journal of Science and Technology*, 9(21), 2016.

[132] Khalifa AlRoum, Abdulhakim Alolama, Rami Kamel, May El Barachi, and Monther Aldwairi. Detecting malware domains: A cyber-threat alarm system. In *International Conference on Emerging Technologies for Developing Countries*, pages 181–191. Springer, 2017.

[133] 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.

[134] Chibuzor John Ugochukwu and EO Bennett. An intrusion detection system using machine learning algorithm. *International Journal of Computer Science and Mathematical Theory*, 4(1):2545–5699, 2018.

[135] Yingsu Qi. Computer real-time location forensics method for network intrusion crimes. *IJ Network Security*, 21(3):530–535, 2019.

[136] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: analysing the rise of iot compromises. In *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*, 2015.

[137] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, pages 57–68. ACM, 2017.

[138] Meng Wang, Javier Santillan, and Fernando Kuipers. Thingpot: an interactive internet-of-things honeypot. *Joint*, 2017.

[139] Peter J Hanson, Lucas Truax, and David D Saranchak. Iot honeynet for military deception and indications and warnings. In *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, volume 10643, page 106431A. International Society for Optics and Photonics, 2018.

[140] Usha Devi Gandhi, Priyan Malarvizhi Kumar, R Varatharajan, Gunasekaran Manogaran, Revathi Sundarasekar, and Shreyas Kadu. Hiotpot: surveillance on iot devices against recent threats. *Wireless personal communications*, pages 1–16, 2018.

[141] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. *Black Hat*, 2017.

[142] Rajesh Kumar Shrivastava, Bazila Bashir, and Chittaranjan Hota. Attack detection and forensics using honeypot in iot environment. In *International Conference on Distributed Computing and Internet Technology*, pages 402–409. Springer, 2019.

[143] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Minh Hoang Dang, N Asokan, and Ahmad-Reza Sadeghi. D\" iot: A crowdsourced self-learning approach for detecting compromised iot devices. *arXiv preprint arXiv:1804.07474*, 2018.

[144] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot-network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[145] Mario Galluscio, Nataliia Neshenko, Elias Bou-Hard, Yongliang Huang, Nasir Ghani, Jorge Crichigno, and Georges Kaddoum. A first empirical look on internet-scale exploitations of iot devices. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*, pages 1–7. IEEE, 2017.

[146] Marco Antonio Sotelo Monge, Andrés Herranz González, Borja Lorenzo Fernández, Diego Maestre Vidal, Guillermo Rius García, and Jorge Maestre Vidal. Traffic-flow analysis for source-side ddos recognition on 5g environments. *Journal of Network and Computer Applications*, 136:114–131, 2019.

[147] Ahmad W Al-Dabbagh, Yuzhe Li, and Tongwen Chen. An intrusion detection system for cyber attacks in wireless networked control systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(8):1049–1053, 2018.

[148] Jonathan Roux, Eric Alata, Guillaume Auriol, Vincent Nicomette, and Mohamed Kaâniche. Toward an intrusion detection approach for iot based on radio communications profiling. In *13th European Dependable Computing Conference*, page 4p, 2017.

[149] Nalam Venkata Abhishek, Teng Joon Lim, Biplab Sikdar, and Anshoo Tandon. An intrusion detection system for detecting compromised gateways in clustered iot networks. In *2018 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 1–6. IEEE, 2018.

[150] Kai Yang, Jie Ren, Yanqiao Zhu, and Weiyi Zhang. Active learning for wireless iot intrusion detection. *arXiv preprint arXiv:1808.01412*, 2018.

[151] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal*, 2018.

[152] Mohammed Saber, Ilhame El Farissi, Sara Chadli, Mohamed Emharraf, and Mohammed Ghaouth Belkasmi. Performance analysis of an intrusion detection systems based of artificial neural network. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pages 511–521. Springer, 2017.

[153] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Christos Tachtatzis, and Robert Atkinson. Shallow and deep networks intrusion detection system: A taxonomy and survey. *arXiv preprint arXiv:1701.02145*, 2017.

[154] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.

[155] Quamar Niyaz, Weiqing Sun, and Ahmad Y Javaid. A deep learning based ddos detection system in software-defined networking (sdn). *arXiv preprint arXiv:1611.07400*, 2016.

[156] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *arXiv preprint arXiv:1709.02656*, 2017.

[157] 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.

[158] 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.

[159] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, 2018.

[160] Mohammad Wazid, Ashok Kumar Das, Neeraj Kumar, and Athanasios V Vasilakos. Design of secure key management and user authentication scheme for fog computing services. *Future Generation Computer Systems*, 91:475–492, 2019.

[161] Srinivas Jangirala, Ashok Kumar Das, and Athanasios V Vasilakos. Designing secure lightweight blockchain-enabled rfid-based authentication protocol for supply chains in 5g mobile edge computing environment. *IEEE Transactions on Industrial Informatics*, 2019.

[162] Harsha Vasudev, Debasis Das, and Athanasios V Vasilakos. Secure message propagation protocols for iovs communication components. *Computers & Electrical Engineering*, 82:106555, 2020.

[163] Feng Chen, Pan Deng, Jiafu Wan, Daqiang Zhang, Athanasios V Vasilakos, and Xiaohui Rong. Data mining for the internet of things: literature review and challenges. *International Journal of Distributed Sensor Networks*, 11(8):431047, 2015.

[164] Mauro Conti, Ali Dehghantanha, Katrin Franke, and Steve Watson. Internet of things security and forensics: Challenges and opportunities. *Future Generation Computer Systems*, 78:544–546, 2018.

[165] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. Towards automation of vulnerability and exploitation identification in iiot networks. In *2018 IEEE International Conference on Industrial Internet (ICII)*, pages 139–145. IEEE, 2018.

[166] Gabriel Arquelau Pimenta Rodrigues, Robson de Oliveira Albuquerque, Flávio Elias Gomes de Deus, et al. Cybersecurity and network forensics: Analysis of malicious traffic towards a honeynet with deep packet inspection. *Applied Sciences*, 7(10):1082, 2017.

[167] Nour Moustafa. A systemic iot-fog-cloud architecture for big-data analytics and cyber security systems: A review of fog computing. *arXiv preprint arXiv:1906.01055*, 2019.

[168] Chang Liu, Chi Yang, Xuyun Zhang, and Jinjun Chen. External integrity verification for outsourced big data in cloud and iot: A big picture. *Future generation computer systems*, 49:58–67, 2015.

[169] Nour Moustafa and Jill Slay. A network forensic scheme using correntropy-variation for attack detection. In *IFIP International Conference on Digital Forensics*, pages 225–239. Springer, 2018.

[170] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Pierre-Louis Dubouilh, Ephraim Iorkyase, Christos Tachtatzis, and Robert Atkinson. Threat analysis of iot networks using artificial neural network intrusion detection system. In *Networks, Computers and Communications (ISNCC), 2016 International Symposium on*, pages 1–6. IEEE, 2016.

[171] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[172] Nour Moustafa, Gideon Creech, Elena Sitnikova, and Marwa Keshk. Collaborative anomaly detection framework for handling big data of cloud computing. In *2017 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6. IEEE, 2017.

[173] Nour Moustafa, Kim-Kwang Raymond Choo, Ibrahim Radwan, and Seyit Camtepe. Outlier dirichlet mixture mechanism: Adversarial statistical learning for anomaly detection in the fog. *IEEE Transactions on Information Forensics and Security*, 2019.

[174] Kun Wang, Miao Du, Yanfei Sun, Alexey Vinel, and Yan Zhang. Attack detection and distributed forensics in machine-to-machine networks. *IEEE Network*, 30(6):49–55, 2016.

[175] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[176] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[177] Nour Moustafa, Gideon Creech, and Jill Slay. Flow aggregator module for analysing network traffic. In *Progress in Computing, Analytics and Networking*, pages 19–29. Springer, 2018.

[178] Olivier De Vel, Alison Anderson, Malcolm Corney, and George Mohay. Mining e-mail content for author identification forensics. *ACM Sigmod Record*, 30(4):55–64, 2001.

[179] Esraa Alomari, Selvakumar Manickam, BB Gupta, Parminder Singh, and Mohammed Anbar. Design, deployment and use of http-based botnet (hbb) testbed. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pages 1265–1269. IEEE, 2014.

[180] Sajal Bhatia, Desmond Schmidt, George Mohay, and Alan Tickle. A framework for generating realistic traffic for distributed denial-of-service attacks and flash events. *Computers & Security*, 40:95–107, 2014.

[181] Sunny Behal and Krishan Kumar. Detection of ddos attacks and flash events using information theory metrics–an empirical investigation. *Computer Communications*, 103:18–28, 2017.

[182] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *Proceeding of the International Conference on Telecommunication, Power Analysis and Computing Techniques, Chennai: IN*, 2017.

[183] Nour Moustafa. New generations of internet of things datasets for cybersecurity applications based machine learning: Ton_iot datasets.

[184] S Terry Brugger and Jedidiah Chow. An assessment of the darpa ids evaluation dataset using snort. *UCDAVIS department of Computer Science*, 1(2007):22, 2007.

[185] Gabriel Maciá Fernández, José Camacho, Roberto Magán-Carrión, Pedro Garcıa-Teodoro, and Roberto Theron. Ugr'16: A new dataset for the evaluation of cyclostationarity-based network idss.

[186] Monowar H Bhuyan, Dhruba K Bhattacharyya, and Jugal K Kalita. Towards generating real-life datasets for network intrusion detection. *IJ Network Security*, 17(6):683–701, 2015.

[187] Center of applied internet data analysis.

[188] Lawrence berkley national laboratory (lbnl), icsi, lbnl/icsi enterprise tracing project, 2005.

[189] University of new Brunswick Canadian Institute of Cybersecurity. Iscx dataset.

[190] Adel Ammar. A decision tree classifier for intrusion detection priority tagging. *Journal of Computer and Communications*, 3(04):52, 2015.

[191] Argus (audit record generation and utilization system).

[192] Esxi hypervisor.

[193] vsphere client.

[194] Iot hub aws.

[195] Mosquitto mqtt broker.

[196] Cron scheduling package.

[197] Swati Paliwal and Ravindra Gupta. Denial-of-service, probing & remote to user (r2l) attack detection using genetic algorithm. *International Journal of Computer Applications*, 60(19):57–62, 2012.

[198] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Understanding passive and active service discovery (extended). Technical report, Technical Report ISI-TR-2007-642, USC/Information Sciences Institute, 2007.

[199] Nazrul Hoque, Monowar H Bhuyan, Ram Charan Baishya, Dhruba K Bhattacharyya, and Jugal K Kalita. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40:307–324, 2014.

[200] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning.* Insecure, 2009.

[201] Xprobe2.

[202] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[203] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[204] A Jesudoss and N Subramaniam. A survey on authentication attacks and countermeasures in a distributed environment. *Indian J Comput Sci Eng IJCSE*, 5:71–77, 2014.

[205] Logkeys software.

[206] Hydra software.

[207] Yun Zheng and Chee Keong Kwoh. A feature subset selection method based on high-dimensional mutual information. *Entropy*, 13(4):860–901, 2011.

[208] David Meyer and FH Technikum Wien. Support vector machines. *R News*, 1(3):23–26, 2001.

[209] Stephen Grossberg. Recurrent neural networks. *Scholarpedia*, 8(2):1888, 2013.

[210] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[211] Jeroen Pijpker and Harald Vranken. The role of internet service providers in botnet mitigation. In *2016 European Intelligence and Security Informatics Conference (EISIC)*, pages 24–31. IEEE, 2016.

[212] Sebastián García, Vojtěch Uhlíř, and Martin Rehak. Identifying and modeling botnet c&c behaviors. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, page 1. ACM, 2014.

[213] Nour Moustafa and Jill Slay. A hybrid feature selection for network intrusion detection systems: Central points. *arXiv preprint arXiv:1707.05505*, 2017.

[214] James Wyke. What is zeus? *Sophos, May*, 2011.

[215] Nickolaos Koroniotis, Nour Moustafa, and Elena Sitnikova. A new network forensic framework based on deep learning for internet of things networks: A particle deep framework. *Future Generation Computer Systems*, 2020.

[216] Kanubhai Patel and Bharat Buddhadev. Predictive rule discovery for network intrusion detection. In *Intelligent Distributed Computing*, pages 287–298. Springer, 2015.

[217] Crina Grosan and Ajith Abraham. Rule-based expert systems. In *Intelligent Systems*, pages 149–185. Springer, 2011.

[218] Ashkan Rahimian, Raha Ziarati, Stere Preda, and Mourad Debbabi. On the reverse engineering of the citadel botnet. In *International Symposium on Foundations and Practice of Security*, pages 408–425. Springer, 2013.

[219] Amir Houmansadr and Nikita Borisov. Botmosaic: Collaborative network watermark for the detection of irc-based botnets. *Journal of Systems and Software*, 86(3):707–715, 2013.

[220] Nathan Goodman. A survey of advances in botnet technologies. *arXiv preprint arXiv:1702.01132*, 2017.

[221] Xiao-Jing Wang and Xiao-yin Wang. Topology-assisted deterministic packet marking for ip traceback. *The Journal of China Universities of Posts and Telecommunications*, 17(2):116–121, 2010.

[222] P Banu Prakash and ES Phalguna Krishna. Achieving high accuracy in an attack-path reconstruction in marking on demand scheme. *i-Manager's Journal on Information Technology*, 5(3):24, 2016.

[223] Kuan-Cheng Lin, Sih-Yang Chen, and Jason C Hung. Botnet detection using support vector machines with artificial fish swarm algorithm. *Journal of Applied Mathematics*, 2014, 2014.

[224] Julie Greensmith. Securing the internet of things with responsive artificial immune systems. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 113–120. ACM, 2015.

[225] Networkminer tool.

[226] Bro tool.

[227] Snort tool.

[228] Nmap tool.

[229] Xplico tool.

[230] Pcapxray tool.

[231] Prads tool.

[232] Tcpstat tool.

[233] Anshul Tayal, Nishchol Mishra, and Sanjeev Sharma. Active monitoring & post-mortem forensic analysis of network threats: A survey. *International Journal of Electronics and Information Engineering*, 6(1):49–59, 2017.

[234] K Nandha Kumar and S Sukumaran. A survey on network intrusion detection system techniques. *International Journal of Advanced Technology and Engineering Exploration*, 5(47):385–393, 2018.

[235] Saad Alabdulsalam, Kevin Schaefer, Tahar Kechadi, and Nhien-An Le-Khac. Internet of things forensics–challenges and a case study. In *IFIP International Conference on Digital Forensics*, pages 35–48. Springer, 2018.

[236] R. C. Joshi and Emmanuel S. Pilli. *Network Forensics*, pages 3–16. Springer London, London, 2016.

[237] Nour Moustafa, Gideon Creech, and Jill Slay. Big data analytics for intrusion detection system: Statistical decision-making using finite dirichlet mixture models. In *Data analytics and decision support for cybersecurity*, pages 127–156. Springer, 2017.

[238] Nguyen Thanh Van, Tran Ngoc Thinh, and Le Thanh Sach. An anomaly-based network intrusion detection system using deep learning. In *2017 International Conference on System Science and Engineering (ICSSE)*, pages 210–214. IEEE, 2017.

[239] Olivier Brun, Yonghua Yin, and Erol Gelenbe. Deep learning with dense random neural network for detecting attacks against iot-connected home environments. *Procedia computer science*, 134:458–463, 2018.

[240] Farha Ali. Ip spoofing. *The Internet Protocol Journal*, 10(4):1–9, 2007.

[241] Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis, and Stefanos Gritzalis. Dns amplification attack revisited. *Computers & Security*, 39:475–485, 2013.

[242] Michal Kováčik, Michal Kajan, and Martin Žádník. Detecting ip spoofing by modelling history of ip address entry points. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 73–83. Springer, 2013.

[243] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[244] Danny Roobaert, Grigoris Karakoulas, and Nitesh V Chawla. Information gain, correlation and support vector machines. In *Feature extraction*, pages 463–470. Springer, 2006.

[245] Nathaniel FG Martin and James W England. *Mathematical theory of entropy*, volume 12. Cambridge university press, 2011.

[246] Lior Rokach and Oded Maimon. Decision trees. In *Data mining and knowledge discovery handbook*, pages 165–192. Springer, 2005.

[247] Weka tool, August 2017.

[248] Fabrice Guillet and Howard J Hamilton. *Quality measures in data mining*, volume 43. Springer, 2007.

[249] Nour Moustafa and Jill Slay. The significant features of the unsw-nb15 and the kdd99 data sets for network intrusion detection systems. In *2015 4th international workshop on building analysis datasets and gathering experience returns for security (BADGERS)*, pages 25–31. IEEE, 2015.

[250] Michael A Berry and Gordon S Linoff. Mastering data mining: The art and science of customer relationship management. *Industrial Management & Data Systems*, 2000.

[251] Hetal Bhavsar and Amit Ganatra. A comparative study of training algorithms for supervised machine learning. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(4):2231–2307, 2012.

[252] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 2005.

[253] Kudakwashe Zvarevashe, Innocent Mapanga, and Prudence Kadebu. A technical evaluation of the performance of classical artificial intelligence (ai) and methods based on computational intelligence (ci) ie supervised learning, unsupervised learning and ensemble algorithms in intrusion detection systems. 2016.

[254] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[255] Liyuan Xiao, Yetian Chen, and Carl K Chang. Bayesian model averaging of bayesian network classifiers for intrusion detection. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 128–133. IEEE, 2014.

[256] Sonepat Area and Ranchi Mesra. Analysis of bayes, neural network and tree classifier of classification technique in data mining using weka. 2012.

[257] Yuming Hua, Junhai Guo, and Hua Zhao. Deep belief networks and deep learning. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pages 1–4, Jan 2015.

[258] Jaehoon Koo and Diego Klabjan. Improved classification based on deep belief networks. *arXiv preprint arXiv:1804.09812*, 2018.

[259] albertbup. A python implementation of deep belief networks built upon numpy and tensorflow with scikit-learn compatibility, 2017.

[260] Junjie Wu. Cluster analysis and k-means clustering: an introduction. In *Advances in K-means Clustering*, pages 1–16. Springer, 2012.

[261] Xin Jin and Jiawei Han. *Expectation Maximization Clustering*, pages 382–383. Springer US, Boston, MA, 2010.

[262] Frank Dellaert. The expectation maximization algorithm. Technical report, Georgia Institute of Technology, 2002.

[263] N Moustaf and Jill Slay. Creating novel features to anomaly network detection using darpa-2009 data set. In *Proceedings of the 14th European Conference on Cyber Warfare and Security. Academic Conferences Limited*, pages 204–212, 2015.

[264] Frans Botes, Louise Leenen, and Retha De La Harpe. Ant colony induced decision trees for intrusion detection. In *16th European Conference on Cyber Warfare and Security*, pages 53–62. ACPI, 2017.

[265] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 2019.

[266] Nour Moustafa and Jill Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, 25(1-3):18–31, 2016.

[267] Adetunmbi A Olusola, Adeola S Oladele, and Daramola O Abosede. Analysis of kdd'99 intrusion detection dataset for selection of relevance features. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, pages 20–22. Citeseer, 2010.

[268] Rong Du, Paolo Santi, Ming Xiao, Athanasios V Vasilakos, and Carlo Fischione. The sensable city: A survey on the deployment and management for smart city monitoring. *IEEE Communications Surveys & Tutorials*, 21(2):1533–1560, 2018.

[269] Mohammad Wazid, Ashok Kumar Das, Neeraj Kumar, Athanasios V Vasilakos, and Joel JPC Rodrigues. Design and analysis of secure lightweight remote user authentication and key agreement scheme in internet of drones deployment. *IEEE Internet of Things Journal*, 6(2):3572–3584, 2018.

[270] Nickolaos Koroniotis, Nour Moustafa, and Elena Sitnikova. Forensics and deep learning mechanisms for botnets in internet of things: A survey of challenges and solutions. *IEEE Access*, 7:61764–61785, 2019.

[271] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of things (iot) communication protocols. In *2017 8th International conference on information technology (ICIT)*, pages 685–690. IEEE, 2017.

[272] S Prabakaran and Shilpa Mitra. Survey of analysis of crime detection techniques using data mining and machine learning. In *Journal of Physics: Conference Series*, volume 1000, page 012046. IOP Publishing, 2018.

[273] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 24, 2015.

[274] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In *ICML 2018 AutoML Workshop*, July 2018.

[275] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.

[276] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems*, pages 3389–3400, 2018.

[277] Jiazhuo Wang, Jason Xu, and Xuejun Wang. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. *arXiv preprint arXiv:1801.01596*, 2018.

[278] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. Hyper-power: Power-and memory-constrained hyper-parameter optimization for neural networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 19–24. IEEE, 2018.

[279] Steve Watson and Ali Dehghantanha. Digital forensics: the missing piece of the internet of things promise. *Computer Fraud & Security*, 2016(6):5–8, 2016.

[280] Maxim Chernyshev, Sherali Zeadally, Zubair Baig, and Andrew Woodward. Internet of things forensics: The need, process models, and open issues. *IT Professional*, 20(3):40–49, 2018.

[281] D Paul Joseph and Jasmine Norman. An analysis of digital forensics in cyber security. In *First International Conference on Artificial Intelligence and Cognitive Computing*, pages 701–708. Springer, 2019.

[282] Duc-Phong Le, Huasong Meng, Le Su, Sze Ling Yeo, and Vrizlynn Thing. Biff: A blockchain-based iot forensics framework with identity privacy. In *TENCON 2018-2018 IEEE Region 10 Conference*, pages 2372–2377. IEEE, 2018.

[283] Aleksandar Valjarevic and Hein S Venter. A comprehensive and harmonized digital forensic investigation process model. *Journal of forensic sciences*, 60(6):1467–1483, 2015.

[284] Luca Caviglione, Steffen Wendzel, and Wojciech Mazurczyk. The future of digital forensics: Challenges and the road ahead. *IEEE Security & Privacy*, 15(6):12–17, 2017.

[285] Seung-Woo Han, Hyunsoo Kwon, Changhee Hahn, Dongyoug Koo, and Junbeom Hur. A survey on mitm and its countermeasures in the tls handshake protocol. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 724–729. IEEE, 2016.

[286] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A Selcuk Uluagac. Iotdots: A digital forensics framework for smart environments. *arXiv preprint arXiv:1809.00745*, 2018.

[287] Niandong Liao, Shengfeng Tian, and Tinghua Wang. Network forensics based on fuzzy logic and expert system. *Computer Communications*, 32(17):1881–1892, 2009.

[288] Abdulghani Ali Ahmed and Mohammed Falah Mohammed. Sairf: A similarity approach for attack intention recognition using fuzzy min-max neural network. *Journal of Computational Science*, 25:467–473, 2018.

[289] Anton Yudhana, Imam Riadi, and Faizin Ridho. Ddos classification using neural network and naïve bayes methods for network forensics. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 9(11):177–183, 2018.

[290] Khoa Nguyen, Dat Tran, Wanli Ma, and Dharmendra Sharma. An approach to detect network attacks applied for network forensics. In *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 655–660. IEEE, 2014.

[291] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Transactions on Sustainable Computing*, 4(1):88–95, 2018.

[292] Xiaoyong Yuan, Chuanhuang Li, and Xiaolin Li. Deepdefense: identifying ddos attack via deep learning. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8. IEEE, 2017.

[293] Quan Le, Oisín Boydell, Brian Mac Namee, and Mark Scanlon. Deep learning at the shallow end: Malware classification for non-domain experts. *Digital Investigation*, 26:S118–S126, 2018.

[294] Ibrahim Alrashdi, Ali Alqazzaz, Esam Aloufi, Raed Alharthi, Mohamed Zohdy, and Hua Ming. Ad-iot: anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0305–0310. IEEE, 2019.

[295] Sajad Homayoun, Marzieh Ahmadzadeh, Sattar Hashemi, Ali Dehghantanha, and Raouf Khayami. Botshark: A deep learning approach for botnet traffic detection. In *Cyber Threat Intelligence*, pages 137–153. Springer, 2018.

[296] Gonzalo De La Torre, Paul Rad, and Kim-Kwang Raymond Choo. Implementation of deep packet inspection in smart grids and industrial internet of things: Challenges and opportunities. *Journal of Network and Computer Applications*, 2019.

[297] James Kennedy and Russell Eberhart. Particle swarm optimization (pso). In *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948, 1995.

[298] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft Computing*, 22(2):387–408, 2018.

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

[300] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[301] Russell C Eberhart and Yuhui Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 94–100. IEEE, 2001.

[302] A Nikabadi and M Ebadzadeh. Particle swarm optimization algorithms with adaptive inertia weight: A survey of the state of the art and a novel method. *IEEE journal of evolutionary computation*, 2008.

[303] Konstantinos E Parsopoulos. Particle swarm methods. *Handbook of Heuristics*, pages 1–47, 2016.

[304] Russ Eberhart, Pat Simpson, and Roy Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., 1996.

[305] Maurice Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1951–1957. IEEE, 1999.

[306] Russ C Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, volume 1, pages 84–88. IEEE, 2000.

[307] James Kennedy and Russell C Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, volume 5, pages 4104–4108. IEEE, 1997.

[308] Rui Mendes, James Kennedy, and José Neves. The fully informed particle swarm: simpler, maybe better. *IEEE transactions on evolutionary computation*, 8(3):204–210, 2004.

[309] Frans Van den Bergh and Andries Petrus Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE transactions on evolutionary computation*, 8(3):225–239, 2004.

[310] Wireshark tool.

[311] Tcpdump tool.

[312] Ettercap tool.

[313] Frederik Armknecht and Andreas Dewald. Privacy-preserving email forensics. *Digital Investigation*, 14:S127–S136, 2015.

[314] Xingguo Lu and Ming Liu. A fuzzy logic controller tuned with pso for delta robot trajectory control. In *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, pages 004345–004351. IEEE, 2015.

[315] Mohd Nadhir Ab Wahab, Samia Nefti-Meziani, and Adham Atyabi. A comprehensive review of swarm optimization algorithms. *PloS one*, 10(5):e0122827, 2015.

[316] Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: a review, 2017.

[317] Micael Couceiro and Pedram Ghamisi. *Particle Swarm Optimization*, pages 1–10. Springer International Publishing, Cham, 2016.

[318] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788, 2018.

[319] Manel Houimli, Laid Kahloul, and Sihem Benaoun. Formal specification, verification and evaluation of the mqtt protocol in the internet of things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pages 214–221. IEEE, 2017.

[320] Antonio Celesti, Davide Mulfari, Maria Fazio, Massimo Villari, and Antonio Puliafito. Exploring container virtualization in iot clouds. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6. IEEE, 2016.

[321] Marc Claesen, Jaak Simm, Dusan Popovic, and BD Moor. Hyperparameter tuning in python using optunity. In *Proceedings of the International Workshop on Technical Computing for Machine Learning and Mathematical Engineering*, volume 1, page 3, 2014.

[322] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[323] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[324] Hamed HaddadPajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo. A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Generation Computer Systems*, 85:88–96, 2018.

[325] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[326] Andysah Putera Utama Siahaan. Intrusion detection system in network forensic analysis and investigation. 2017.

[327] Eric Conrad, Seth Misenar, and Joshua Feldman. Chapter 7 - domain 7: Security operations. In Eric Conrad, Seth Misenar, and Joshua Feldman, editors, *Eleventh Hour CISSPÂ® (Third Edition)*, pages 145 – 183. Syngress, third edition edition, 2017.

[328] B. B. Gupta and Omkar P. Badve. Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment. *Neural Computing and Applications*, 28(12):3655–3682, Dec 2017.

[329] Thijs Rozekrans, Matthijs Mekking, and Javy de Koning. Defending against dns reflection amplification attacks. *University of Amsterdam System & Network Engineering RP1*, 2013.

[330] Douglas C MacFarland, Craig A Shue, and Andrew J Kalafut. The best bang for the byte: Characterizing the potential of dns amplification attacks. *Computer Networks*, 116:12–21, 2017.

[331] Opeyemi A Osanaiye. Short paper: Ip spoofing detection for preventing ddos attack in cloud computing. In *2015 18th International Conference on Intelligence in Next Generation Networks*, pages 139–141. IEEE, 2015.