



Final Paper

CM3203 – One Semester Individual Project – 40 Credits

**Evaluation of Traditional Intrusion Detection Systems in the
Internet of Things**

Author: Haya Mohammed Alsultan

Supervisor: Dr. Philipp Reinecke

Moderator: Dr. Charith Perera

A Final Year Project Submitted for the Degree of Bachelor of Science

Department of Computer Science with Security and Forensics

Cardiff University

2019

Abstract

Internet of Things (IoT) is a new paradigm that integrates the Internet and physical objects belonging to different domains such as home automation, industrial processes, human health and environmental monitoring. It deepens the presence of Internet-connected devices in our daily activities, bringing, in addition to many benefits, challenges related to security issues. Intrusion Detection Systems (IDS) have been an important tool for the protection of networks and information systems. Traditional IDSs are designed to operate in traditional networks. Due to the particular characteristics of IoT devices, such as constrained-resource devices, specific protocol stacks, and standards, applying traditional IDSs to protect these devices might not be suitable. In this report, the accuracy and capability of two traditional IDSs, Snort and Suricata, were evaluated in IoT. Several types of attacks were implemented in order to observe the ability and extent to which the IDSs can detect attacks. The main finding of this project was that traditional IDSs were able to detect the majority of attacks in IoT when attack specific rules were identified. However, the main obstacle was enabling the IDS to see the traffic of the IoT device. This process indicated that traditional IDSs may not be suitable for deployment in IoT.

Acknowledgments

I would like to take this opportunity to express my gratitude to the people who have supported me throughout this project.

I would first like to thank my supervisor, Dr. Philipp Reinecke, for his guidance and support throughout the semester. It was a pleasure working with him.

I would also like to thank my parents, Asma and Mohammed, for always believing in me. Another thank you to my sisters and brother for their constant support.

Finally, a special thank you to my friends who were always there for me.

Table of Contents

1. INTRODUCTION	1
1.1 Preface	1
1.2 Project Aims and Scope	2
1.3 Intended Audience.....	3
1.4 Report Structure	3
2. BACKGROUND	4
2.1 Internet of Things.....	4
2.1.1 Differences between IoT and Traditional network	6
2.2 Security Challenges in IoT.....	6
2.3 Intrusion Detection Systems	7
2.3.1 Snort	9
2.3.2 Suricata	10
2.3.3 Bro	11
2.4 Attacks	13
2.4.1 Port Scanning.....	13
2.4.2 Man in the Middle.....	14
2.4.3 Denial of Service	15
2.5 Related Work	16
2.5.1 Evaluation of IDS in Traditional Network	16
2.5.2 Intrusion Detection in IoT.....	19
3. EVALUATION APPROACH AND DESIGN	21
3.1 Evaluation Metrics	21
3.2 Scenario's Design	22
3.2.1 Traffic.....	23
3.3 Overview of Attack Generation Tools.....	23
3.4 Limitations of Approach.....	25
4. IMPLEMENTATION OF EVALUATION.....	26
4.1 Evaluation Environment	26
4.1.1 Attacks Generation.....	29
4.1.2 IoT Device Setup	33
4.1.3 Installation of Snort.....	34

4.1.4 Installation of Suricata.....	35
4.2 Rules	36
4.2.1 DoS Rules	37
4.2.2 Port Scanning Rules	38
5. DISCUSSION AND RESULTS.....	39
5.1 First Scenario Results	39
5.2 Second Scenario Results	42
5.3 Third Scenario Results	45
5.4 Summary of Results	47
6. FINAL THOUGHTS	49
6.1 Conclusion.....	49
6.2 Future Work	50
6.3 Reflection on Learning	51

List of Figures

Figure 1. The IoT five-layer architecture [22].	5
Figure 2. A comparison of the traditional TCP/IP stack used by most Internet hosts and the corresponding protocols used in an IoT network, based on [12].	6
Figure 3. Basic Process of IDS [35].	8
Figure 4. The architecture of Snort [2].	10
Figure 5. The architecture of Suricata [12].	11
Figure 6. The architecture of Bro [44].	12
Figure 8. Setup of Scenario 1.	27
Figure 9. Setup of Scenario 2.	27
Figure 10. Setup of Scenario 3	28
Figure 11. Ping Test Rule	35
Figure 12. Snort's Ping Alert	35
Figure 13. Suricata's Ping Alert.....	36
Figure 14. Example Rule [40].	37
Figure 15. Results of Snort Capability with Heavy Traffic	48

List of Tables

Table 1. A comparison of Snort, Suricata, and Bro (adapted from [20]).....	12
Table 2. IDS Evaluation Metrics used by Different Research.	18
Table 3. Evaluation Metrics.	22
Table 4. Summary of Scenarios.	22
Table 5. Summary of Attacks Generated.	29
Table 6. Snort Results in Scenario 1, Experiment 1.....	39
Table 7. Suricata Results in Scenario 1, Experiment 1.	40
Table 8. Snort Results in Scenario 1, Experiment 2.....	41
Table 9. Suricata Results in Scenario 1, Experiment 2	41
Table 10. Snort Results in Scenario 2, Experiment 1.....	42
Table 11. Suricata Results in Scenario 2, Experiment 1	43
Table 12. Snort Results in Scenario 2, Experiment 2.....	44
Table 13. Suricata Results in Scenario 2, Experiment 2	44
Table 14. Snort Results in Scenario 3, Experiment 1.....	45
Table 15. Suricata Results in Scenario 3, Experiment 1	46
Table 16. Snort Results in Scenario 3, Experiment 2.....	46
Table 17. Suricata Results in Scenario 3, Experiment 2	47

Table of Abbreviations

IoT	Internet of Things
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
MITM	Man in the Middle
DOS	Denial of Service
DDoS	Distributed A Denial of Service
TLS	Transport Layer Security

1. Introduction

This chapter will provide a preface to the report, describe the project aims and scope, identify the intended audience, and present the report structure.

1.1 Preface

Internet of Things (IoT) is a network of computing devices that are embedded in physical objects which are interconnected using the internet, enabling these devices to send and receive data [8]. It is a paradigm that integrates physical objects and the Internet. It combines connectivity with devices, sensors, and people, allowing a form of conversation between human and machine, software and hardware [14]. These conversations can enable devices to predict, react, and improve the physical world similarly to how the internet currently uses networks and computer screens to improve the information world [14]. One of the main aims of IoT is to enhance the quality of human life in terms of comfort and efficiency. It allows individuals to automate, achieve and control the tasks that are essential for life and provides better responses as a result [9]. It is applied in several domains, such as the automation of homes and environmental monitoring. It is estimated that by 2020, the world will have approximately 30 billion connected devices. It is also estimated that the economic value of IoT will approximately be \$1.46 trillion in 2020 [14]. Although the potential for IoT is vast, its practical execution still remains in its beginnings.

Security issues are introduced with the wide spread of IoT as a result of vulnerabilities in the infrastructure of such devices, which could affect its applications [1]. This recent popularity of IoT devices and the significant increase in their usage provides cybercriminals with additional opportunities to perform malicious acts against individuals or organisations with IoT capabilities and creates a potential attack vector [1]. Many tools are deployed to reduce or overcome such security issues, for instance, Intrusion Detection Systems (IDS). IDSs have been an important tool for the protection of information systems and networks. An IDS is a system that attempts to achieve detection of intrusive acts by comparing noticeable behaviour against suspicious patterns in network resources and detecting irregular behaviour and abuses. IDSs are usually created to operate in a traditional network setting [1]. One of the main characterises of IoT devices is that they have limited computing capabilities and storage

capacity. Also, IoT networks use protocols that are not employed in traditional networks. Due to these particular characteristics, applying traditional IDSs may not be suitable for IoT environments and devices [43].

1.2 Project Aims and Scope

The main aim of the project was to evaluate the accuracy and capability of a traditional Intrusion Detection System in an Internet of Things environment. The overall objective was to understand whether a typical user could deploy a traditional IDS, as they are widely accessible and easily usable, to detect intrusions against their IoT device and their suitability in an IoT environment. In order to achieve this, a number of sub-goals were established:

- Identify evaluation criteria in terms of their relevance to the evaluation in order to develop a systematic way of evaluating the applicability of the different IDSs to IoT environments.
- Conduct the evaluation and observe the results.
- Identify any differences in the behaviour of the traditional IDSs in IoT versus in a traditional setting and explore potential reasons for the differences.
- Offer an informed and educated analysis of how a traditional IDS could be applied to IoT.

The scope of this project was to evaluate two intrusion detection systems in an IoT environment, Snort and Suricata, which were chosen due to their flexible deployment and popularity. A smart plug was used as the IoT device. The evaluation was conducted in terms of two main measures, accuracy and capability. The accuracy measured the ability and extent of the system to detect intrusive activities, whereas the capability was measured by the IDS's ability to handle the network traffic, without dropping packets. To evaluate the IDSs against these measures, three categories of attacks were implemented, Port Scanning, Man in The Middle attacks, and Denial of Service attacks. The results of these tests were presented and discussed. The purpose was to identify if, and to what extent, IDSs which have been designed for traditional environments would work in IoT, where differences in behaviour were observed and potential causes for any differences were explored.

1.3 Intended Audience

The intended audience and beneficiaries from this project are individuals who are interested or doing research in the field of the security of IoT devices and would like to utilize an intrusion detection system in order to detect any malicious activities occurring against their IoT device.

1.4 Report Structure

This report is organised in the following way: **Chapter 2** presents an initial background of Internet of Things, Intrusion Detection Systems, and types of attacks. Additionally, research related to this project is discussed in this chapter. **Chapter 3** discusses the approach taken to perform the evaluation. **Chapter 4** focuses on the execution of the evaluation. **Chapter 5** presents the results of the evaluation, summarises the significant findings, concludes the project, and discusses the potential future work that could be undertaken to improve the project. Finally, **Chapter 6** reflects on the learning obtained from undertaking this project.

2. Background

This chapter will briefly explain several concepts that were essential to understand in order to conduct the evaluation. Firstly, it will discuss the concept of Internet of Things, how it differs from a traditional network, and its security related challenges. Secondly, an introduction into Intrusion Detection Systems and possible types of attacks is presented. Finally, related work is discussed in the context of its relevance to the project.

2.1 Internet of Things

The term “Internet of Things” was first used in 1999 by British technology pioneer Kevin Ashton to explain the concept of a system where physical objects could be connected to the Internet by sensors [32]. The main idea of IoT is to establish an integration between the physical and the cyber worlds. Applications of IoT can vary from a simple appliance for a smart home to a sophisticated equipment for an industrial plant. Despite the different purposes of IoT applications, they still share some common characteristics [12].

There is no single IoT architecture that is agreed on universally and shared between all IoT devices. The most basic and commonly used architecture is a three-layer architecture, which was presented in the early stages of research in this area. The following summary of the architecture is based on [43][32][9]. The architecture has three layers, namely, the perception layer, network layer, and application layer. The perception layer is the physical layer, where the collection phase begins. In this layer, data is gathered about the physical environment using a combination of technologies and devices for sensing short range communication. These devices are usually small in size, have limited CPU, memory, and restricted power resources. Sensors are used for identifying and gathering the information, such as physical parameters or identifying other smart objects in the environment. There are many types of sensors attached to objects to collect this information, such as RFID and 2-D barcodes, where the sensors are chosen according to the purpose of each application. The network layer, also known as transmission layer, acts like a bridge between perception layer and application layer. It is responsible for transmitting the information collected from the physical objects through sensors to applications and, subsequently, the user. In this transmission phase, a network is built using certain technologies to interconnect users and objects across longer distances, where the medium for the transmission can be wireless or wire based. Examples of the technologies used include Wi-Fi, Ethernet, Digital Subscriber Line (DSL), Hybrid Fibre Coaxial (HFC), and

combining these technologies with TCP/IP protocols. This layer is also responsible for connecting the smart objects, network devices, and different networks to one another. The application layer is responsible for delivering application specific services to the user. It defines various applications in which IoT can be deployed, for example, smart homes, smart cities, and smart health. The gathered information about the physical environment is used to possibly take decisions such as controlling the physical objects to act on the physical environment. Additionally, the use of middleware is included in this layer, as it is responsible for enabling the communication and integration between the different physical objects and multi-platform applications [43][32][9].

According to Sethi [32], the three-layer architecture played an important role in the development of IoT, however, it also brought along a set of specific security and storage challenges. Hence, researchers proposed a five-layer architecture to make the IoT more secure, which is illustrated in figure 1. It has three layers, a perception layer, transport layer, and application layer, that is similar to the architecture described above. However, it also has two additional layers, namely, a processing layer and a business layer [32]. The processing layer, also known as a middleware layer, collects the information that is sent from a transport layer and performs processing onto the collected information. It has the responsibility to eliminate extra information that has no meaning and extracts the useful information. Furthermore, the business layer refers to an intended behaviour of an application and acts like a manager of a whole system. It has a number of responsibilities that include the management and control of applications, management of the user's privacy, and the ability to determine how information can be created, stored and changed [32].

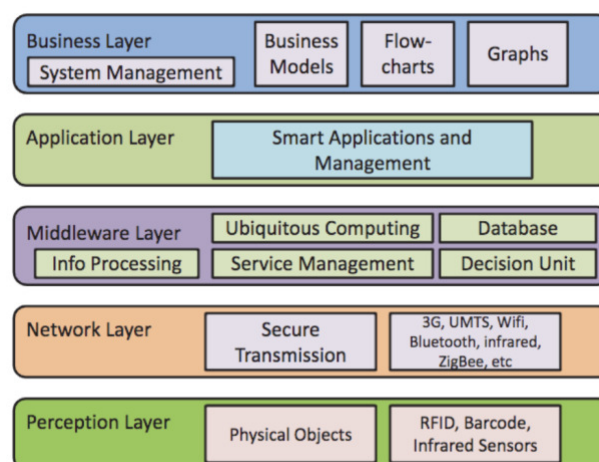


Figure 1. The IoT five-layer architecture [22].

2.1.1 Differences between IoT and Traditional network

In order to evaluate how security measures designed for traditional environments might behave in IoT settings, an analysis of the differences between the two environments is outlined below.

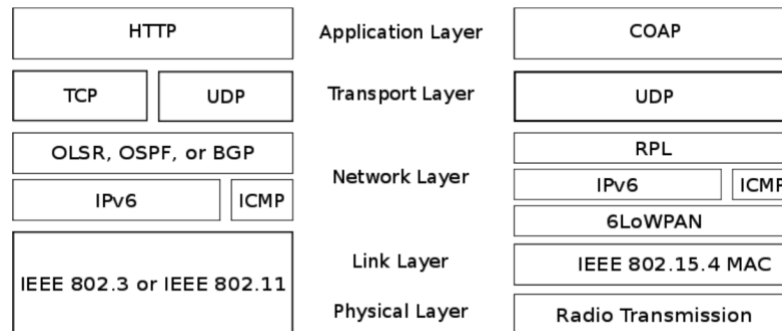


Figure 2. A comparison of the traditional TCP/IP stack used by most Internet hosts and the corresponding protocols used in an IoT network, based on [12].

IoT devices typically connect to the Internet through the IP (Internet Protocol) stack [32]. This stack is very complicated and requires a large amount of power and memory from the connecting devices. The IoT devices can also connect locally through non-IP networks, which consume less power, and connect to the Internet via a smart gateway. Non-IP communication channels such as Bluetooth, RFID, and NFC are limited in their range; hence, their applications are limited to small personal area networks. Personal area networks (PAN) are widely used in IoT applications such as wearables connected to smartphones [32]. In order to increase the range of such local networks, modification of the IP stack was needed in order to facilitate low power communication using the IP stack. One of the solutions is 6LoWPAN, which incorporates IPv6 with low power personal area networks. The range of a PAN with 6LoWPAN is similar to local area networks, and the power consumption is much lower. The leading communication technologies used in the IoT are IEEE 802.15.4, low power WiFi, 6LoWPAN, RFID, NFC, Sigfox, LoraWAN, and other proprietary protocols for wireless networks [32].

2.2 Security Challenges in IoT

A significant amount of technologies for IoT have been proposed by different organization, consortiums, or special interest groups, which could pose a great challenge for the end-to-end security of IoT applications [43]. The security of these applications is a serious issue as it could affect many individuals due to this increasing number of services and users in IoT networks.

Security challenges in these systems are related to the security issues arising in the different IoT layers [43]. As discussed by Burhan [9], common security threats of the perception layer include eavesdropping, replay attack, timing attacks. The network layer is vulnerable to attacks targeting integrity and authentication of information that is being transferred in the network. Attacks of this nature include Man in The Middle attacks, Denial of Service attacks, sniffing, gateway attacks, and unauthorized access [9]. A number of security issues can arise in the application layer. When IoT is used to create a smart home for instance, it introduces many threats and vulnerabilities from the inside and outside. To implement strong security in an IoT based smart home, one of the main issues is that the devices used in smart homes have weak computational power and a low amount of storage. Malicious code attacks, application vulnerabilities, and software bugs are security challenges that could be faced in this layer [9][12]. As IoT networks are vulnerable to attacks such as the ones mentioned above, a line of defence, designed for detecting these attacks is needed. Intrusion Detection Systems can fulfil this purpose [43].

2.3 Intrusion Detection Systems

There are several security mechanisms usually deployed by individuals or organisations regarding physical infrastructure, such as antiviruses between the client and server, firewalls, and SPAM-filters. The next level of security in systems is the use of an intrusion detection system (IDS).

Intrusion detection is the activity of detecting actions that intruders execute against information systems [43]. The main aim of attacks is usually to gain unauthorized access to a certain computer system. Intruders can either be external, such as users that are outside the targeted network trying to gain unauthorized access to the information of a system, or internal, such as users inside a network that try to increase their limited access privileges to unauthorized privileges with intentions of misuse [43]. Therefore, an IDS is a system that attempts to accomplish intrusion detection by comparing suspicious patterns against noticeable behaviour. It accomplishes this by monitoring network resources and detecting anomalous and irregular behaviours and misuses [1].

As illustrated in figure 3, the basic process of an IDS involves three main components, sensors, an analysis engine, and a reporting system [43]. Firstly, sensors are used for data collection and are placed at different network places or hosts. They monitor the network and collect data such

as packet headers, traffic statistics, changes in the file-system, service requests, and operating system calls [43]. Afterwards, the sensors forward the collected data to the analysis engine, which is responsible for detection, as it examines the collected data to detect any ongoing intrusions, symptoms of attacks, or other policy violations [43]. There are two main approaches that the analysis engine can use individually or concurrently to perform the analysis [20]. The first approach is Misuse/Signature-Based Detection, where this type of detection engine detects intrusions that comply with well-known attack signatures or patterns that target known software vulnerabilities. The main limitation of this approach is that it only looks for the known weaknesses and may not consider the detection of unknown future intrusions [20]. The second approach is Anomaly/Statistical Detection, where this detection engine will search for something abnormal or anomalous by analysing system event streams, using statistical techniques to find patterns of activity that seem to be unusual. Then, if the analysis engine detects an intrusion, the reporting system generates an alert to the network administrator [20].

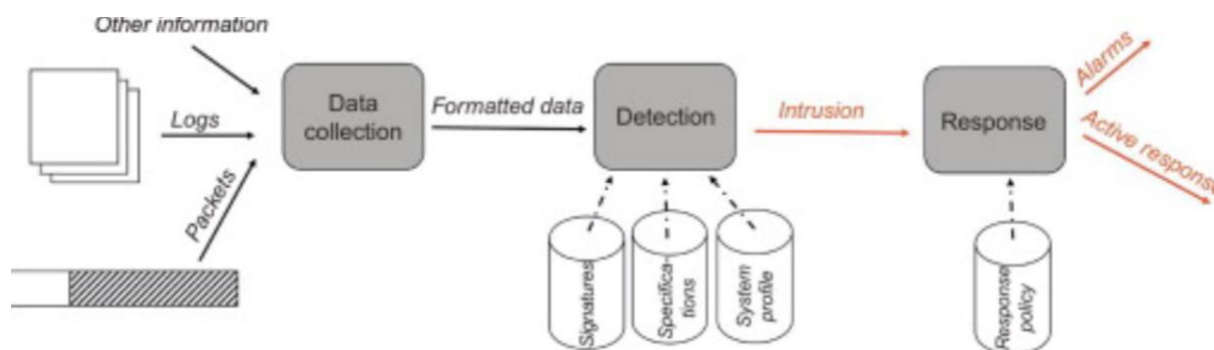


Figure 3. Basic Process of IDS [35].

There are two main classifications of IDSs, Network-based Intrusion Detection System (NIDS) and Host-based Intrusion Detection System (HIDS). NIDS, which can either be a software-based system or a hardware-based system, monitors network traffic packets to detect intrusions and malicious attacks, while operating on stand-alone devices on a network. It performs an analysis of passing traffic on the entire subnet and matches the traffic that is passed to the library of known attacks. Once an attack is identified, or abnormal behaviour is sensed, an alert can be sent to the administrator. On the other hand, Host-based Intrusion Detection System (HIDS) operate on a single workstation in order to monitor traffic on its host machine by utilizing the resources of its host to detect attacks, such as the log files in a computer system. Unlike NIDS, the HIDS analyses not only network traffic but also system calls, running processes, file-system changes, and application logs [43][29].

There are a number of NIDSs and HIDSs available for use. For the purpose of this project, three NIDSs were considered for evaluation, which were Snort [36], Suricata [37], and Bro [44].

2.3.1 Snort

The first IDS considered in this project was Snort [36]. Snort was developed in 1998 by Martin Roesch [1]. It is a lightweight and open source signature-based IDS that can be used for multiple purposes as it allows several configuration modes, for instance, sniffer mode, packet logger mode, or NIDS or NIPS mode. When Snort is in sniffer mode, it will read network packets and then show them in a continuous stream on the Snort console [1]. If packet logger is selected, Snort will log the read packets to the disk in the default output format i.e. ASCII text. When Snort is used as a NIDS, it will read the network traffic and analyse the traffic against a defined ruleset in order to detect any malicious attacks. It can rely on customized user rules or signatures sourced rules from databases like Emerging Threats [1]. For the purpose of this project, Snort will be configured as a NIDS. Snort has been developed for several operating systems, such as Linux and Windows, to detect any incoming threats. Both signature-based intrusion detection as well as anomaly-based methods are used by Snort. It consists of a number of logically separated components that serve different roles, as demonstrated in figure 4 [1]. These components work in synchronization to identify attacks and produce output in a required format:

- The first component is a packet decoder, which takes packets from network interfaces and prepares them to be pre-processed or to be sent to the detection engine.
- The second component is a pre-processor that acts as a plug-in with Snort to coordinate or modify data packets before sending them to the detection engine to discover if the packet is being used by an intruder, to regulate protocol headers, identify irregularities, packet reassembly, and to perform TCP stream reassembly.
- The third component is a detection engine, which is responsible for detecting if any intrusion activity exists in a packet using Snort rules, where these rules are compared against all packets. If a packet matches any rule, predefined action specified in the rule is taken else the packet is dropped.
- The fourth component is a logging and alerting system that generates alert messages depending on the findings of the detection engine.

- The fifth component is the output module that processes alerts and logs and generate final result for the user to access them in ways as needed, whether it was from the console, external files, or databases [39].

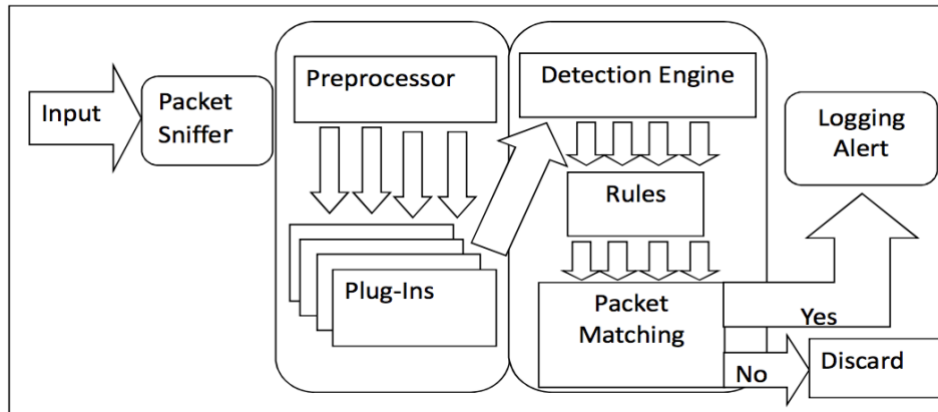


Figure 4. The architecture of Snort [2].

2.3.2 Suricata

The other IDS considered was Suricata [37]. It is a signature-based network IDS developed by the Open Information Security Foundation (OISF). As other IDSs, Suricata monitors network traffic and generates alert logs when malicious traffic is detected. Suricata is designed to be compatible with other security components. It has a number of features, such as unified output functionality, and its ability to accept calls from other applications through its pluggable libraries. Suricata is compatible with most operating systems, such as Linux and Windows [31]. There are many similarities between Suricata and Snort. For example, they offer the same operation modes, they have similar ways to connect to networks, and the general data flow through Suricata is similar to Snort as packets are captured, decoded, processed and then analysed. Additionally, Suricata also uses rulesets to perform detection that have similar syntax to rulesets used by Snort. The most commonly used are Emerging Threats, Emerging Threats Pro and Sourcefire's VRT [31]. However, Suricata differs when it comes to the internals of its engine. It uses a multi-threaded approach of detection using a number of thread modules. Each thread has an input queue handler and an output queue handler that are used to acquire packets from other threads, or from the global packet pool. As this multi-threading technique is deployed, multiple-packet captures queues are allowed for each worker process, which distributes the workload across multiple CPU cores available [31]. Suricata's multi-threaded architecture is illustrated in figure 5. Suricata also features the HTP Library that is a HTTP normalizer and parser written by Ivan Ristic for the OISF that integrates and provides advanced

processing of HTTP streams for Suricata. The HTP library is required by the engine but can also be used as an independent tool. Moreover, while Snort works in the network and transport layer, Suricata has the additional ability to work at the application layer, which improves its malwares detection ability [11].

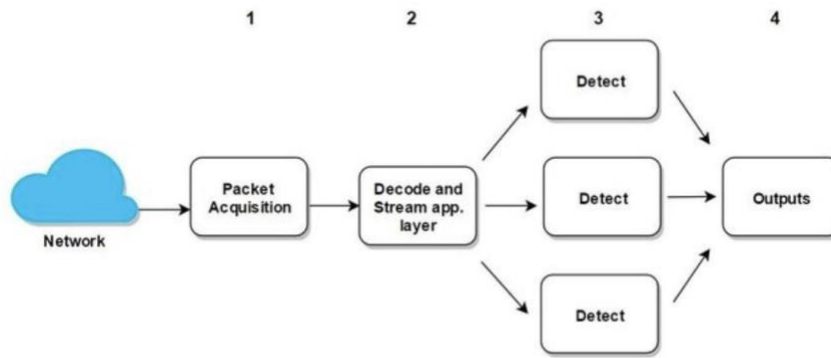


Figure 5. The architecture of Suricata [12].

2.3.3 Bro

Bro was also considered as it is one of the most well-known IDSs [44]. Bro is a passive, open-source network traffic analyser. Its primary use is to act as a security monitor that examines all traffic on a connection in detail for signs of suspicious activity. The Bro IDS is an anomaly-based intrusion detection system. Bro is a domain-specific language for networking applications in which Bro IDS is written as this policy engine has its own language [1].

Bro IDS has a number of key components that include libpcap, an event engine, and a policy script interpreter [1]. Using the libpcap library, Bro is able to capture packets from network interfaces, handle all traffic that is arriving from the network layer, and eliminate non-important elements. Afterwards, the filtered packet stream is forwarded to the Event engine. The received packets are combined together in order to take necessary actions by the event engine, as it is responsible for reducing the incoming packet stream into a series of higher-level events. These events demonstrate network activity in policy-neutral terms that describe what has been seen and identifies significant parts of the connection [1]. For example, every HTTP request on the wire transforms to a corresponding `http_request` event that describes the involved IP addresses and ports, the URI being requested, and the HTTP version in use. However, other interpretations are not conveyed any further by the event, for instance, whether that URI corresponds to a known malware site [1]. On the other hand, Bro's third main

component provides such semantics. The script interpreter implements a collection of event handlers written in Bro's custom scripting language. It also matches the packets with the rules, which leads to the finding of suspicious and dangerous actions and ignores unmatched packets. Bro scripts can generate real-time alerts [1]. As rules in Bro work with scripts and is a script driven IDS, its approach to detection is considered significantly different when compared to the other tools. It distributes the load to the multiple servers, which supports high throughput environments [1]. The internal architecture of Bro is demonstrated below in figure 6:

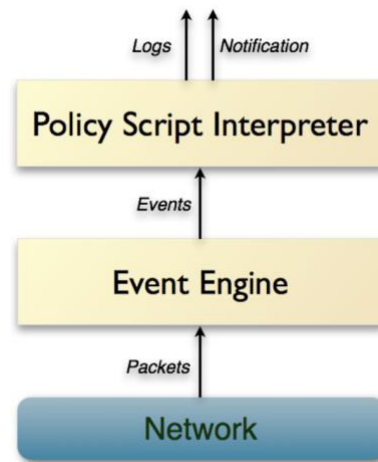


Figure 6. The architecture of Bro [44].

The following table illustrates a comparison of the three IDSs described above.

Parameter	Snort	Suricata	Bro
Supported Platform	Win, MacOS, Unix	Win, MacOS, Unix	MacOS, Unix
Support to high speed network	Medium	High	High
Rules	VRT Snort rules Emerging Threats rules	VRT Snort rules Emerging Threats rules	Pre-packaged scripts
Offline Analysis	Yes, for multiple files	Yes, for single file	Yes, for single file
Threads	Single thread	Multithreaded	Single thread
IPv6 Support	Yes	Yes	No
Installation and Deployment	Easy	Easy	Difficult

Table 1. A comparison of Snort, Suricata, and Bro (adapted from [20]).

The IDSs that were chosen to be evaluated in this project were Snort and Suricata. These two were chosen due to their popularity among security tools and their feasible deployment. As both of these IDSs are widely used and accessible, a user may choose to deploy them to protect their IoT device, which is a reason for why evaluating their accuracy in such an environment would be important. The Bro IDS was not evaluated in this project as it is considered relatively difficult to interpret and configure. During this project, several attempts to install Bro were conducted, however, it was shown that the process was complex for a beginner and infeasible due to the lack of documentation and community support it has. Its users are expected to have a certain level of programming knowledge and experience with the tool in order to execute its rules. Because of this, evaluating it in both a traditional setting and an IoT setting was infeasible.

2.4 Attacks

The security of IoT devices could be compromised through a variety of attacks. Each layer in an IoT device is subject to a number of these attacks, that could target the device's confidentiality, integrity, and availability. These attacks include Port Scanning, Man in the Middle, and Denial of Service attacks, which were implemented in this project in order to evaluate the IDSs in IoT. This section will discuss the nature of these attacks and their relevancy to the evaluation.

2.4.1 Port Scanning

A port scan is a method for determining which ports on a network are open. As ports on a computer are the place where information is sent and received, running a port scan on a network or server reveals which ports are open, listening, or receiving information [11]. It can be utilized to reveal the presence of security devices such as firewalls that are present between the sender and the target. Due to this functionality, it is a popular reconnaissance tool for attackers seeking a weak point of access to break into a computer. It is also valuable for testing network security and the strength of the system's firewall. Applications that perform port scanning are designed to connect to a wide range of ports or IP addresses on a network, a single IP address, or a specific list of ports and IP addresses by sending carefully prepared packets to each destination port number [11]. Four types of port scanning attacks were considered in this project, Connect Scans, FIN Scans, TCP Scans, and XMAS Scans. It is important to evaluate an IDS in IoT against port scanning as attackers might apply this type of attacks to identify

open ports in an IoT device's network to perform malicious acts against the device, hence, why an IDS should be able to detect such an attack and alert the user.

2.4.2 Man in the Middle

Man-in-the-Middle (MITM) is an active eavesdropping attack. MITM attacks are one of the main and oldest forms of conducting a cyber-attack. It poses a serious threat to security as it provides the attacker with the ability to capture and manipulate sensitive information in real-time [18]. A MITM attack occurs when an attacker intercepts communication between two parties either to secretly eavesdrop or modify traffic traveling between the two [38]. MITM has a broad range of techniques and potential outcomes, depending on the target and the goal. It can be deployed to achieve a number of goals, varying from wanting to steal personal information or login credentials, pry on the victim, disrupt communications, or damage data [38]. Ways attackers often carry out this type of attack include Address Resolution Protocol (ARP) spoofing, Domain Name System (DNS) spoofing, Spanning Tree Protocol (STP) mangling, and Internet Control Message Protocol (ICMP) redirection. An attacker can also exploit vulnerabilities in a wireless router's security configuration caused by weak or default passwords [18]. For example, a malicious router, also called an evil twin, can mirror legitimate Wi-Fi access points that will be entirely controlled by these malicious actors, who can now monitor, collect or manipulate all information the user sends. Damages caused by MITM can vary in significance, depending on the attacker's goals and ability to cause disruption [18]. In order to prevent against attacks of MITM nature, forms of endpoint authentication are now incorporated in most cryptographic protocols. For example, the Transport Layer Security (TLS) protocol can be required to authenticate one or both parties using a mutually trusted certification authority[18].

Due to the lack of proper security implementation in IoT devices, it makes them more vulnerable to MITM attacks as standard mitigations against this type of attack are not implemented [19]. For instance, many IoT devices do not implement the TLS protocol used to prevent this type of attack or implement older versions of it that are not as strong. Also, due to IoT device's potential to deliver a large amount of personally identifying information about individuals and businesses and their ability to act as a gateway to these sensitive and personal information, it makes them particularly an appealing prospect for cybercriminals to hijack their traffic [19]. An example that demonstrates the effect of this type of attack on IoT devices is the finding of a MITM vulnerability in the Samsung smart fridge [24]. While performing penetration testing, security

researchers have discovered a potential way to steal users' Gmail credentials from the smart fridge. Although the fridge implemented a Secure Socket Layer protocol, it still failed to validate SSL certificates, thereby enabling MITM attacks against most connections [24]. Nowadays, the majority of people use the same passwords for most personal accounts, therefore, this type of attack could provide a hacker access to multiple account with just using a single piece of information [24]. For the purpose of this project, an ARP Spoofing attack was conducted.

2.4.3 Denial of Service

A Denial of Service (DoS) attack is an attack that is launched to make the resources of a particular network or system unavailable and inaccessible for the legitimate user [41]. DoS attacks to networks are frequent and potentially harmful. In this type of attack, attackers target the availability of a network, which is easily achievable as administrative privileges are not required by the target network or system. Most DoS attacks exploit vulnerabilities in TCP/IP stack protocols. Attackers either use a single computer to launch these attacks or multiple computers, where the usage of multiple computers is known as a Distributed Denial of Service (DDoS) attack. Detection of DoS attacks can be challenging as attackers can be situated anywhere across the world, which makes it difficult to differentiate the attacks from legitimate traffic [41]. DoS attacks can be divided into three main types, volume-based attacks, protocol attacks, and application layer attacks [17]. In volume-based attacks, the goal is to saturate the bandwidth of the target network. Examples of this type of attack are UDP flooding and ICMP flooding. Moreover, in protocol attacks, the aim is to consume server resources, or intermediate communication equipment, such as firewalls and load balancers. Examples include SYN floods, fragmented packet attacks, Ping of Death, and Smurf DDoS. Application layer attacks are comprised of seemingly legitimate and innocent requests that could result in the halt of web servers. For instance, low-and-slow attacks, GET/POST floods, or attacks that target Apache vulnerabilities [17].

IoT is a popular target for such attacks. IoT devices can also be used to perform such attacks, for example, the Mirai botnet [3]. In 2016, this botnet was able to temporarily disable a number of high-profile administrations, such as OVH, Dyn, and Krebs on Security via massive distributed DDoS attacks. These attacks were conducted using hundreds of thousands of compromised IoT devices, such as air-quality monitors, personal surveillance cameras, and home routers [3]. A combination of factors contributed to the massive scale of this botnet,

which included an efficient spreading based on Internet-wide scanning, widespread use of insecure default passwords in IoT products, and the insight that keeping the botnet's behaviour simple would allow it to infect many heterogeneous devices [3]. As IoT devices are used in many individuals' daily lives, attacking them by making them unavailable to the legitimate user can have significant repercussions. Hence, the detection of such attacks is essential. In the scope of this project, four DOS type of attacks were considered, SYN Flood, UDP Flood, PUSH ACK Flood, and IP Spoofing.

2.5 Related Work

There has not been any conduction of research that is directly related to the evaluation of a traditional IDS in an IoT environment. However, there is research regarding the evaluation of an IDS in a traditional environment. This research was helpful as it illustrated the different evaluation metrics that can be considered while performing the evaluation in IoT and several evaluation approaches.

2.5.1 Evaluation of IDS in Traditional Network

Several researchers have evaluated traditional IDSs in traditional networks using different evaluation techniques and metrics. A study conducted by Kumar [23] explored various performance metrics used to evaluate IDSs based on standard datasets, where their pros and cons were demonstrated. This study helped with providing a better understanding of the different metrics used for evaluating the performance of the IDSs as it was found that most evaluations were conducted using the detection rate, false positive rate, and area under the ROC (AUC), which is a measure of the ability to correctly classify events as normal or intrusive, as metrics. It was highlighted that these metrics failed to distinguish the performance of the IDSs in some special cases, therefore, a single objective metric called Intrusion Detection Capability (CID) was proposed in this research. Based on this research, the detection rate was chosen as the main metrics in this evaluation.

A research by Belkasmi et al. [6] presented an evaluation approach that is based on a series of tests with the aim of measuring the performance of components of an IDS and their effects on a system, as well as a study on the effect of the characteristics of the deployment environment on the operation of the IDS. This was done by installing Snort IDS on machines with different technical characteristics and using a design of a network to generate a set of experiments to measure the performances obtained in the case of a deployment in high-speed networks. These

experiments consisted of injecting various traffic loads, characterized by different transmission times, packet numbers, packet sizes and bandwidths, and then analysing, for each situation, the processing performed on the packets. This main finding of this research was that Snort had an inability to process multiple packets and the propensity to deposit, without analysis, packets in high-speed networks with heavy traffic. Similar to this research, Snort's and Suricata's capability to process packets based on the average of packets they would drop in each test was evaluated in this project.

A research by Tayyebi et al. [9] also evaluated traditional IDSs' performance based on their processing of packets. In this research, Snort, Suricata, and Bro IDS were analysed and compared for features, general working behaviour, utility, and performance so that individuals or organizations can choose the appropriate tool as per their requirements. One of the findings of this research was that both Bro and Suricata IDSs have the ability to run in high-speed environments. This was demonstrated by their capability to fully capture data from networks without dropping packets, which makes them suitable for more large-scale networks. In contrast, it was found that the performance of Snort in high speed networks was low as it was unable to capture data without dropping packets or slowing down the traffic.

El-Hajj et al. [11] proposed detection approaches in their research article. It was discussed that evaluation can be done using data sets that are available online or by creating a dataset manually. The available data sets do not provide complete logs and are usually outdated, while the simulated data sets provide logs that do not resemble real-life scenarios. Therefore, their research focused on generating port scanning benchmarks that researchers can use to test their detection methods. They suggested a simulation framework using OMNeT++ to generate benchmarks that resemble real-life traffic. As per this related work, the third scenario of this evaluation was conducted by using the IDS in offline mode using a created dataset.

Furthermore, Massicotte et al. [26] elaborated on the different techniques used to test an IDS. Two main evaluation approaches were discussed, the IDS stimulator approach, and the vulnerability exploitation program approach. IDS Stimulators are used for a number of testing purposes such as the generation of false alarms by sending packets that resemble well-known intrusions for cross-testing NIDS signatures and for testing the IDS engine. These tools rely on publicly available signature databases to generate test cases, however, in most situations, the needed signatures are undisclosed or not available from vendors. The vulnerability exploitation approach is then used to overcome this limitation. The use of vulnerability exploitation

programs for IDS testing and evaluation usually implies building a test bed where the attacks are launched using these vulnerability exploitation programs. The attack traffic can be combined with real or emulated normal traffic as background, where the traffic is either recorded for off-line IDS testing or testing the IDS in real-time on the test bed network. Based on this related work, the testing of the IDSs in IoT in the third scenario included the attack traffic being combined with real background traffic during the evaluation.

Table 2 presents a summary of the various evaluation metrics used by several researchers to evaluate IDSs. These metrics were categorised into two main types, efficiency and effectiveness metrics. Efficiency metrics deal with the resources needed to be allocated to the system, including CPU cycles and main memory. Effectiveness metrics represent the ability of the system to distinguish between intrusive and non-intrusive activities.

Research Study	Evaluation Metrics Used	
	Effectiveness	Efficiency
Evaluation Metrics for Intrusion Detection Systems - A Study [23].	False Positive Rate (FRP) Detection Rate (DR)	N/A
Real traffic logs creation for testing intrusion detection systems [11].	The Hit Rates False Alarm Ratios	Robust Performance
Implementation and Performance Evaluation of Intrusion Detection and Response System [6]	N/A	Packets Captured (PCA) Packets Analysed (PAN) Packets Dropped (PDR)
Snort Performance Evaluation [2].	Alert Generation	CPU Usage Packets Dropped (PDR)
A Comparative Analysis of Open-source Intrusion Detection Systems [11].	N/A	CPU Usage Memory Usage Packets Analysed (PAN) Packets Dropped (PDR)
Evaluation of Intrusion Detection Systems in Virtualized Environments [28].	Detection Rate (DR)	CPU Usage Memory Usage

Table 2. IDS Evaluation Metrics used by Different Research.

2.5.2 Intrusion Detection in IoT

Research has been conducted regarding the application of intrusion detection safety measures in IoT environments. The following subsection presents some of this work in a bid to inform the potential differences between IDSs designed for traditional environments versus IoT environments.

A paper by Zarpelaoa [4] discussed the potential reasons for why IDS technologies designed for traditional networks might be inadequate for IoT system, due to their particular characteristics. The first reason related to the processing and storage capacity of network nodes that host IDS agents. In traditional networks, the system administrator deploys IDS agents in nodes with higher computing capacity. As IoT networks are usually composed of nodes with resource constraints, finding nodes with the ability to support IDS agents is more difficult. The second reason related to the network architecture. In traditional networks, end systems are directly connected to specific nodes, which can be done using wireless access points, switches, or routers that are responsible for forwarding the packets to the destination. IoT networks, on the other hand, are usually multi-hop. Then, regular nodes may simultaneously forward packets and work as end systems. This kind of architecture poses new challenges for IDSs. The final reason related to specific network protocols. IoT networks use protocols that are not employed in traditional networks, such as IEEE 802.15.4, IPv6 over Low-power Wireless Personal Area Network (6LoWPAN), IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) and Constrained Application Protocol (CoAP). Different protocols bring original vulnerabilities and new demands for IDS [4].

A paper by Sforzin et al. [33] proposed an intrusion detection architecture for IoT. The feasibility of employing a commodity device, such as Raspberry Pi, as the core component of the architecture was discussed. The performance of the Raspberry Pi while running a modified version of Snort was evaluated. The experiments showed that an architecture based on resource constrained devices can effectively serve as IDS in a distributed system such as IoT [33].

Moreover, Jun and Chi [19] proposed an IDS integrated with complex event-processing (CEP) technology. The benefit of CEP technology is the ability to identify complex patterns via real-time data processing. The event-processing IDS architecture consists of an event filtering unit, an event database unit, a CEP unit and an action engine unit. The system depends on an event-processing model that uses the rule model approach to detect intrusions. The main features of

this system are that it operates in real time and shows high performance in detecting intrusions in an IoT system using an event-processing mechanism [19].

Kasinathan et al. [21] proposed an IDS that detects DoS attacks based on 6LoWPAN in IoT networks. They proposed a DoS detection architecture that consists of an IDS probe, a DoS protection manager and a Suricata IDS. The IDS probe and the IDS were connected using a wired connection via a USB interface. They designed this system based on a study of the vulnerabilities present in IP-based WSNs. The Suricata IDS runs on a host computer; thus, the advantage of this system is that it can overcome the problem of power consumption, leading to conserving power resources in WSNs [21].

As the creation of IDS for IoT is a relatively new area of research, they are not widely accessible and open source. Also, these solutions do not cover a wide range of attacks and IoT technologies. Hence, why a user would attempt to deploy a traditional IDS to protect their IoT device.

3. Evaluation Approach and Design

As illustrated by the related work, there is not a clear and direct way to evaluate an IDS, particularly an IDS in an IoT environment. This chapter of the report will focus on the approach taken to perform the evaluation and how each scenario was designed. The main objective was to evaluate the accuracy and capability of a traditional IDS in an IoT environment.

3.1 Evaluation Metrics

The first part in the process of the evaluation was to establish metrics that the evaluation of the IDSs would be based on. In this project, the evaluation metrics that were used to evaluate Snort and Suricata were accuracy and capability. A summary of these metrics is shown below in table 3. The capability metric will be represented by the average of packets that the IDS would drop when tests are conducted for each type of attack. A dropped packet is a packet that the IDS was not able to analyse when it was run in real-time. The percentage of dropped packets indicates limitations in the capability of an IDS to handle traffic, as when it exceeds its capability, packets will be dropped. Dropped packets also imply false negatives as they are never analysed, thus, any malicious content within them will not be detected. The Detection rate (DR) was used to represent the accuracy of the IDS, which was computed as the ratio between the number of correctly detected attacks and the total number of attacks conducted. A confusion matrix, which represents true and false classification results, was used to produce a numeric value for the detection rate. Each test would be categorised as follows:

- True Positive (TP): Intrusion that are successfully detected by the IDS and provided relevant alerts.
- False positive (FP): Normal/non-intrusive behaviour that was wrongly classified as intrusive by the IDS.
- True Negative (TN) Normal/non-intrusive behaviour was successfully labelled as normal/non-intrusive by the IDS.
- False Negative (FN) Intrusion that were missed by the IDS and classified as normal/non-intrusive.

Then, the detection rate was computed as follows: $DR = \frac{\text{Correctly detected attacks}}{\text{Total number of attacks}} = \frac{TP}{TP+FN}$

Evaluation Metric	Measure	Description
Accuracy	Detection Rate	The ratio between the number of correctly detected attacks and the total number of attacks conducted.
Capability	Packets Dropped	The number and percentage of the packets dropped from the total packets captured.

Table 3. Evaluation Metrics.

3.2 Scenario's Design

This subsection will describe the three different Scenario's that Snort and Suricata were evaluated in and their relevance to the evaluation of a traditional IDS in IoT. Table 4 demonstrates a summary of these scenarios, that will later be described in detail in this section.

Scenario	Description
1	Evaluate IDSs in real-time analysis mode in a traditional setting.
2	Evaluate IDS in real-time analysis mode in an IoT setting.
3	Evaluate IDS in offline mode in an IoT setting with the inclusion of background traffic.

Table 4. Summary of Scenarios.

- The first scenario was to evaluate how each IDS would perform in a traditional setting, that is without the involvement of the IoT device. It was beneficial to conduct this evaluation as it provided an understanding of how the IDS is typically deployed, how to configure it, and what is expect when it is applied to the IoT device. To do this, different attacks were conducted from an attacker machine, where the machine that hosts Snort and Suricata was the target.
- The second scenario was to evaluate Snort and Suricata in an IoT environment in which only the traffic between the IDSs host machine and the IoT device was monitored by the IDSs in real-time. This was done by generating the attacks from the same machine where the IDSs were installed, as that would enable them to see the attack traffic against the IoT

device. However, the problem with this approach to evaluate an IDS in IoT is that it is not realistic in terms of how an IDS would usually work. Realistically, an IDS should be able to monitor all the traffic of the machine it is protecting and not just the attack traffic. Hence, the IDSs were evaluated in a third scenario.

- In the third scenario, Snort and Suricata were evaluated in offline mode using captured traffic of the IoT device and real background traffic. This was done by redirecting any traffic between the device and the internet and the attacker machine to a different machine. Afterwards, this traffic was combined with real background traffic collected from a busy server. The resultant file was then used to run Snort and Suricata in offline mode to present any alerts. This presents a more realistic approach in the sense that IDSs would usually be used with this type of traffic as it contains noise.

3.2.1 Traffic

There were a number of considerations when choosing network traffic for this evaluation. To begin with, many evaluations test IDSs without using any background traffic by using the attack traffic on its own. In such experiments, an IDS is set up on a host or network where there is no activity. Then, computer attacks are launched on this host or network to determine whether or not the IDS can detect the attacks. The former was the approach in scenarios 1 and scenario 2 as testing schemes using this approach are often less costly to implement than the ones that include sanitized or created background traffic and logs. In these scenarios, Snort and Suricata were running simultaneously in all tests to guarantee authentic and fair results. However, this technique can have a disadvantage as it is based on the implicit assumption that an IDS's ability to detect an attack is the same regardless of background activity. To overcome this disadvantage, the IDSs were evaluated in the scenario 3 to provide more realistic results as in this experiment, background traffic was used in combination with the attack traffic.

3.3 Overview of Attack Generation Tools

In order to evaluate Snort and Suricata, a number of attacks were generated to observe if the attacks would be detected or not, where the same attacks were generated in all the scenarios. As mentioned in the background, three main categories of attacks were implemented in the evaluation process, Port Scanning, Man in the Middle, and Denial of Service attacks. A number of penetration testing and monitoring tools were used to generate the attacks and monitor the

traffic, including Nmap, Ettercap, Dsniff, Metasploit, Hping3, and Wireshark. This section will provide brief descriptions of these tools and their use during the evaluation.

Nmap, which stands for Network Mapper, is an open source tool for network discovery and security auditing [30]. Nmap uses raw IP packets in novel ways to determine the hosts that are available on the network, what services, such as application names and versions, those hosts are offering, the operating system and version they are running, and the type of packet filters/firewalls are in use. Even though it was intended to rapidly scan large networks, it performs well against single hosts [30]. Nmap was used to perform all the port scanning attacks to discover vulnerable and open ports on the target machine.

The MITM attack was conducted using two different tools, Ettercap and Dsniff. Ettercap is a comprehensive collection for man in the middle attacks [13]. It features sniffing of live connections, content filtering, supports active and passive dissection of many protocols and includes many features for network and host analysis. It has the feature of ARP poisoning or spoofing to sniff on a switched LAN between two hosts [13]. Using Ettercap, it was possible to choose the target and attacker to conduct a MITM attack and intercept the traffic. Dsniff is a collection of network traffic analysis tools that analyse different application protocols and extract relevant information and interesting data, which can be utilised to perform MITM attacks [10]. Dsniff provided a package that was also used to implement the ARP spoofing attacks.

To implement the DOS attack, the Metasploit framework and Hping3 tool were utilised. The Metasploit Framework is a penetration testing and development platform that grants access to the latest exploit code for various applications, operating systems, and platforms [27]. It is an open source tool based on the concept of 'exploit', meaning that it passes a code that violates the security procedures of a system and attempts to access it. When access is achieved, the Metasploit framework runs a 'payload', which is a code that performs operations on a target machine. Modules are used to provide such functionality to the framework, where each module can serve different purposes. A module can be an exploit, auxiliary, payload, no operation payload (NOP), or post-exploitation module. For example, any module that opens a shell on a target is an exploit module [27]. Hping is a command-line oriented tool utilized to perform analysis and assembly on TCP/IP packets [16]. The interface is inspired by the ping Unix command; however, it also offers different features than just the ability to send ICMP echo

requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, and the ability to send files between a covered channel [16].

Finally, Wireshark was used to capture traffic and generate PCAP files [42]. It is a network analysis tool that captures packets in real time and displays them in human readable format. Mainly, it is a network packet analyser that provides the significant and insignificant details about a network's protocols, decryption, packet information, and more. The information that is retrieved via this tool can be viewed through a GUI or the TTY mode TShark Utility [42]. In the evaluation process, Wireshark was used to monitor the traffic during each attack to confirm that it happened.

3.4 Limitations of Approach

There were a number of limitations to the approach of this project. The first limitation related to the types of IDSs to be evaluated. Initially, the aim of the project was to evaluate Snort and Bro in IoT, where Snort is a signature-based IDS and Bro is an anomaly-based IDS. However, as mentioned in the background section, applying Bro IDS was infeasible. To overcome this, the Suricata IDS was evaluated instead, which is also a signature-based IDS. This limited the variety of the types of IDSs that could have been evaluated in IoT.

Another limitation was that the evaluation process was not automated. The lack of automation was a natural first step because in order to know what to automate, manual tests had to be conducted first. Also, the evaluation was still achieved regardless of the automation. However, this project informs a potential future automation.

Additionally, as the IDSs were not installed on the same IoT device they were protecting in the second and third scenarios, a number of limitations were introduced. The first one was that it provided limitations on the type and amount of traffic the IDSs could monitor in real-time, which could have reduced their detection capabilities. As illustrated in the related work section, a study used an IDS probe to gather the IoT environment's traffic, however, applying this idea was out of scope for this project. Therefore, alternative methods were implemented in order to obtain such traffic, such as using ARP Spoofing. The second limitation was that it affected the variety of evaluation metrics that can be used. For example, using performance as a metric became irrelevant in the second and third scenarios as measuring CPU and memory utilization

of the IDSs on the IoT device was infeasible. However, measuring the capability of the IDSs was still relevant and was conducted in this evaluation.

4. Implementation of Evaluation

The IDSs were evaluated in three different scenarios, where two different experiments were conducted in each scenario. Each IDS was evaluated against numerous attacks, where the evaluation metrics were represented by the detection rate for accuracy and the average of dropped packets for capability. This chapter of the report will discuss the setup of the evaluation environment in each scenario, the generation of attacks process, the setup of the IoT device, the installation of Snort and Suricata, and the development of attack specific rules.

4.1 Evaluation Environment

The first step of the evaluation was to set up an evaluation environment for each scenario. In the first scenario, the IDS was evaluated in a traditional setting, as illustrated by figure 8. This required two machines, an attacker machine and a target machine. The target machine was the machine where Snort and Suricata were installed to detect any attacks, while the attacker machine was the machine used to generate the attacks against the target. For the target machine, an Ubuntu virtual box was installed. Ubuntu was chosen to host the IDSs for several reasons. First, it is user-friendly as its user interface is comprehensible and easy to use. It also has pre-installed packages and tools that supported the process of this evaluation. Additionally, it has strong community support as there are several Linux forums available on the internet that also discuss the use of Snort and Suricata. For the attacker machine, a Kali Linux virtual box was installed. Kali was chosen to generate the attacks in this case as it encompasses several penetration testing tools that do not require an installation process, such as Nmap, the Metasploit framework, Hping3, and Dsniff.

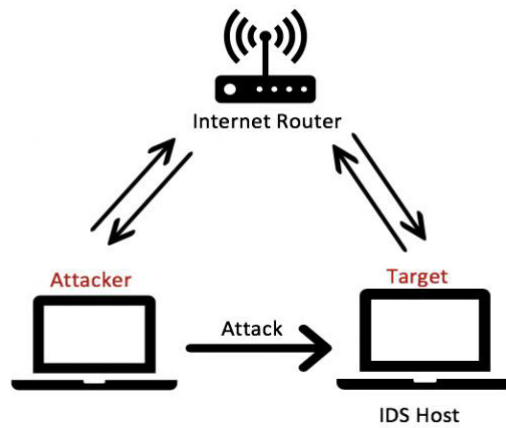


Figure 7. Setup of Scenario 1.

In the second scenario, the IoT element was introduced. The target was the IoT device, while the attacker was the Ubuntu machine, where the IDSs were installed. As indicated in the approach section, the Ubuntu machine was used as the attacker machine to enable the IDSs to view the attack traffic against the IoT device. The IoT device that was used to provide the IoT environment was a TP Link Kasa Smart Plug, which was chosen due to its high usability and popularity. The setup of this scenario is illustrated below in figure 9.

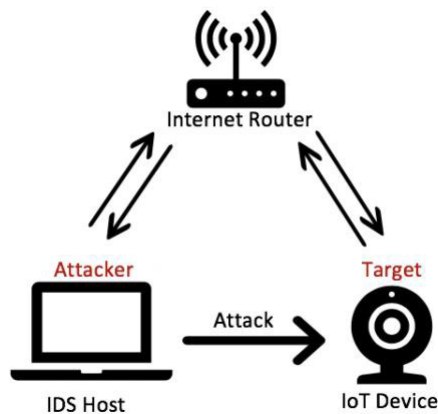


Figure 8. Setup of Scenario 2.

In the third scenario, the target was the IoT device, while the attacker machine was the Kali Linux. This scenario differs from the second one as in this scenario, the traffic of the IoT device was redirected to the Ubuntu machine using ARP Spoofing, which will be discussed in more detail in a following section. After redirecting the traffic of the device to the Ubuntu machine, the attacks were generated using Kali Linux. The attack traffic was then captured along with the traffic between the device and the internet using the Ubuntu machine and combined with

real background traffic collected. The resultant PCAP file was then used to run Snort and Suricata in offline IDS mode to log any attacks contained within it. As discussed in the approach section, this scenario provides a more realistic evaluation of how an IDS would usually operate as it analyses traffic with noise. The setup of this scenario is illustrated below in figure 10.

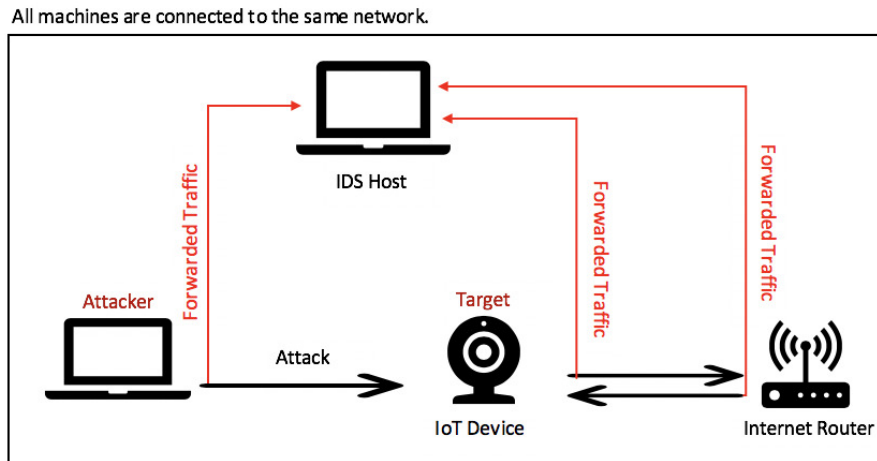


Figure 9. Setup of Scenario 3

As shown in the above figures, in each scenario, a number of attacks were conducted to evaluate the IDSs. The same attacks were conducted in all the scenarios. A summary of these attacks is demonstrated in table 5. In each experiment, each attack was tested 5 times if its results were consistent throughout all tests, and 10 times if they were not, in order to provide further confirmation of the results.

Attack Category	Attack
Port Scanning	Connect Scan
	TCP Scan
	XMAS Scan
	FIN Scan
Denial of Service	SYN Flood
	UDP Flood
	PUSH ACK Flood
	IP Spoofing
Man in the Middle	ARP Spoofing

Table 5. Summary of Attacks Generated.

4.1.1 Attacks Generation

Even though each evaluation was conducted under different circumstances in each scenario, the generation of attacks was the same in all the scenarios. The only part that required changing while generating was the specification of the target IP address of the attack, which differs based on the scenario. Each attack was run several times in order to provide further confirmation of the results and observe any difference in the behaviour of the IDSs with the change of traffic in each iteration of an attack. Prior to performing any attacks, an important step was to begin monitoring the traffic on the Ubuntu machine that hosts the IDSs to confirm that the attacks were implemented successfully and to capture the attack traffic of each test. The reason for capturing the files was to be able to use it in the case of needing to repeat a specific test, which can be done by running Snort and Suricata in offline mode. The specific attacks generated were a Connect scan, FIN scan, TCP scan, XMAS scan, ARP spoofing, SYN flood, UDP flood, PUSH ACK flood, and IP Spoofing. This subsection will discuss the nature of these attacks and how they were generated.

4.1.1.1 Connect Scan

In this type of scan, a connection with all 65,536 ports is established one at a time by sending a request to connect in the form of a SYN flag, and upon receiving an

acknowledgement of connection response in the form of SYN-ACK, it sends back an ACK flag. This SYN, SYN-ACK, ACK exchange comprises a TCP handshake.

In order to conduct the connect scan, the Nmap tool was used by specifying the IP address of the target machine.

4.1.1.2 TCP Scan

In a TCP scan, the three-way handshake is completed, and after a successful attempt, it is logged as a connection. This type of attack does not require root privileges. The port is considered open when the connection is established and closed otherwise. If the connection is successful, the attacker sends a Finish (FIN) packet to end the connection. This type of scan is usually recorded in the log file of the server.

Nmap was used to conduct this attack by specifying -sT and the port to scan, which was 22.

4.1.1.3 Stealth Scan

These types of Scans are of particular interest to an attacker as they are difficult to detect. In these attacks, the packets' flags are manufactured in a way that makes it possible to access the system without having the TCP handshake process to establish a connection. This type of attack includes XMAS scans and FIN scans, which will be detailed below:

- XMAS Scan:

This scan will connect with the target by sending data packets through FIN, PSH & URG flags.

Nmap was used to implement this attack. The XMAS scan is specified by -sX, where the attack destination port was specified as 22 as that will listen for incoming traffic when packets arrive from FIN, PSH & URG flags.

- FIN Scan

In this attack, an unsolicited FIN flag, which is used normally to end an established session, will be sent to a port. The system's response to this random flag can reveal the state of the port or insight about the firewall.

Using Nmap, this scan was implemented, where it was specified by -sF. The attack destination port was specified as 22 as that will listen for packets coming from FIN flags.

4.1.1.4 ARP Spoof

ARP is a protocol used by the data link layer to map an IP address to a MAC address [4]. A host sending a packet needs to know a recipient's MAC address prior to capturing the network layer packet in a data link layer frame. To find the MAC address of a host given its IP address, the source node broadcasts an ARP request packet that asks about the MAC address corresponding to the owner of the IP address. While the ARP request is broadcast, the ARP reply is unicast. As all nodes inside the LAN receive this request, the node that owns this IP address responds with its MAC address. When the host receives the ARP reply, it gets saved in an ARP table which contains entries of mappings of IP addresses and MAC addresses that correspond to one another. This table acts as a cache, where a node will send an ARP request only if the ARP table does not contain the IP/MAC mapping [4]. ARP is a stateless protocol, meaning that it accepts ARP replies without the need to send an ARP request, which makes it relatively weak. An ARP spoofing attack takes advantage of this vulnerability. This attack is conducted by sending ARP reply messages that contain the IP address of a network resource, such as the default gateway or a DNS server, to a victim machine. The attacker replaces the MAC address of the corresponding network resource with his machine's MAC address, where the victim's machine that receives the spoofed ARP replies will not be able to differentiate between fake ones and legitimate ones. Now, any traffic directed to the legitimate resource will be sent through the attacking system, resulting in the attacker playing the role of man in the middle. The attacker is now able to monitor packets, which he can then use to capture sensitive data. The attacker may also modify that data, and then pass it to the designated destination. As this attack occurs on the lower levels of the TCP/IP protocol stack, the end-user is unaware to the attack occurrence [4].

This attack was generated using the Dsniff tool. The first step was to tell the attacker machine using a certain command to collect all traffic on the network. The next step was to use ArpSpoof tool available in the Dsniff collection to specify two IP addresses, for instance, the IP addresses of the router and the target machine, where any traffic between these two was forwarded to the attacker machine.

4.1.1.5 SYN Flood

TCP is a reliable connection that ensures that when data is sent by a particular sender, it will be fully received by the receiver [25]. To establish communication between the sender and the receiver, TCP follows a three-way handshake. A SYN flood attack exploits a weakness in the TCP connection sequence, where the sender is the attacker and the receiver is the victim. In a SYN flood scenario, the attacker sends a SYN packet and the server responds with SYN-ACK, however, the attacker does not reply with an ACK packet. The server expects an ACK packet from the attacker and waits for some time while the attacker continues to send significant amounts of SYN packets, where the server continues to wait for the final ACK until timeout. This will result in the server exhausting its resources waiting for ACK [25].

In order to implement this SYN flood attack, the Metasploit framework was used. It contains an auxiliary module that is able to implement a variety of DoS attacks, with a section that focuses on TCP related DoS attacks, particularly SYN Flood, which was acquired. The next step was to specify the target IP address by setting the rhost and specifying the attacker IP address by setting the shost. Using the Nmap connect scan described above, vulnerable ports were identified and then specified by setting the rport, which was set to port 80. Afterwards, the attack was conducted using the exploit command, which sent countless SYN packets to the target machine to demolish its services.

4.1.1.6 UDP Flood

The goal of this attack is to flood random ports on the target machine with significant numbers of User Datagram Protocol (UDP) packets [25]. When the target gets a packet on a port, it looks out for an application that is listening to that port. When it does not find the packet, it replies back with an Internet Control Message Protocol (ICMP) packet. ICMP packets are used to send error messages, such as 'Destination Unreachable'. When a lot of UDP packets are received, the target consumes a lot of resources in replying back with ICMP packets. This can prevent the target from responding to legitimate requests. This process overwhelms host resources, which can ultimately lead to inaccessibility [25].

To implement this attack, the HPing3 tool was used. Numerous bits of packets per second were sent to the target machine on port 80.

4.1.1.7 PUSH ACK Flood

When connecting with a server, a client can ask for confirmation that the information was received by setting the ACK flag, or it can force the server to process the information in the packet by setting the PUSH flag. Both requests require the server to do more work than with other types of requests. In a PUSH ACK Flood attack, the server was flooded with illegitimate PUSH and ACK requests, which can prevent the server from responding to valid traffic. Since PUSH and ACK messages are a part of standard traffic flow, a significant amount of these messages alone indicates abuse.

The Hping3 tool was also used to implement this attack, where a combination these flags was used to disrupt the services of the target by generating an endless number of PSH-ACK packets. The number of packets to be sent each second was set to 10000 along with the flags to flood with, -PA, and a destination port of 80.

4.1.1.8 IP Spoofing

In this type of attack, the attacker tries to impersonate an IP address so that they can pretend to be another user. During this attack, the attacker machine sent numerous packets from false and different source address to the target machine in order to overwhelm the target.

The Hping3 tool was used to conduct this attack. The number of packets to be sent each second was set to 10000 using -c, while the size of packets was set to 120 using -d. The type of packets to send was SYN packets and the TCP window size was set 64. The destination port of the attack was chosen as 21 for the FTP port. To hide the IP address of the attacker, --rand-source was used which will use random source IP addresses to flood the target. Finally, the target machine IP address was specified.

4.1.2 IoT Device Setup

The process of setting up the Smart Plug for the second and third scenarios went as follows, the plug has two physical buttons: An on/off relay switch and a device reset button that resets the device if pushed for five seconds or longer. When plugged in, an unconfigured or freshly reset Smart Plug started an unsecured open Access Point with the SSID “TP-LINK_Smart Plug_XXXX” where XXXX are four hexadecimal numbers. TP-Link’s Smart Home mobile application “Kasa” made the smartphone connect to this access point, then, it sent UDP broadcast packets to 255.255.255.255 to find the Smart Plug IP and proceeded to configure it

with the SSID and password that was entered into the app. The Smart Plug then turned off the Access Point and connected to the configured WiFi as a client.

Using the Linux netdiscover tool, all IP addresses of the machines connected on a certain network were displayed, which confirmed that all the involved machines were connected to the same network.

4.1.3 Installation of Snort

As part of setting up the evaluation environment, Snort was installed on the Ubuntu machine. It was installed following the Snort official documentation and guides available from Snort's official website [36]. This process consisted of a number of steps. Before installing Snort, a number of software packages and pre-requisites had to be installed, including pcap (libpcap), PCRE (libpcre3), Libdnet, and DAQ. Libpcap is used by monitoring software such as Snort to capture packets traveling over a network, save these captured packets to a file, and read files containing saved packets, where Snort uses these files to read network traffic and analyse it. Libdnet is a generic networking API that provides access to several protocols. All these Snort pre-requisites are available from the Ubuntu repositories. The latest version of the Data Acquisition Library (DAQ) can be downloaded from the Snort website. It was introduced in the latest version of Snort for packet inputs and outputs. The DAQ replaces direct calls to libpcap functions with an abstraction layer that facilitates operation on a variety of hardware and software interfaces without requiring changes to Snort. It is possible to select the DAQ type and mode when invoking Snort to perform pcap readback or inline operation. Snort can be set up using its source code, which is also available on the Snort website, where options of different versions are available.

The next step was configuring Snort for the particular system used in order to run it in NIDS mode, which included editing some configuration files. The following step was to download and maintain the detection rules that Snort will follow to identify potential threats and have the latest detection capabilities. Snort provides three types of rule sets, community rules, registered rules, and subscriber rules. Community rules are freely available; however, they are slightly limited. The second option is to register for free on Snort's website and get access to an Oink code, which allows downloading of registered users rule sets that have more variety. Lastly, subscriber rules are the ones available to users with an active subscription to Snort services. For the purpose of this project, the registered user's ruleset will be used as it represents what

the average user of Snort would usually download. Snort generates alerts according to the rules defined in configuration file. Sourcefire Vulnerability Research Team (VRT) Rules are the official rules of snort.org. Each rule is developed and tested using the same standards the VRT uses for its customers. In order to ensure that rules are constantly updated with the latest VRT rules, PulledPork was installed. PulledPork is an open source Perl script that is used to manage VRT Rules updates. Also, Barnyard, which is an output system for Snort, was installed. Snort creates a special binary output format called unified. Barnyard reads this file, and then resends the data to a database back-end.

Snort allows for customisation of rules, therefore, in my many cases, certain rules will have to be created and specified by the user in the local.rules file for Snort to be able to detect these particular attacks. In order to make sure that Snort is able to read the rules and act upon them, a test rule was created that alerts the user when an attempt to ping the machine occurs, where each time an ICMP ping request happened, the user would be alerted. figure 11 and figure 12 below demonstrate the rule used and a sample result of testing. The \$EXTERNAL_NET represents the external network and the \$HOME_NET represents the home network that is being protected.

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP test"; sid:10000001; rev:001;)
```

Figure 10. Ping Test Rule

```
04/23-17:23:25.883036  [**] [1:10000001:1] ICMP test [**] [Priority: 0] {ICMP} 192.168.1.185 -> 192.168.1.151
```

Figure 11. Snort's Ping Alert

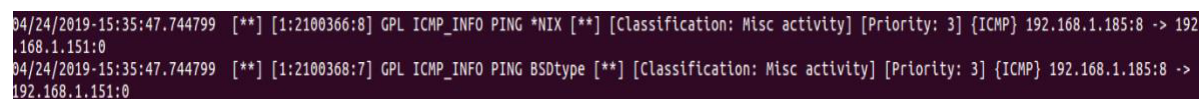
When Snort detects and alerts about a particular attack, it captures the packets that indicated a malicious act and resulted in an alert in a PCAP file, which can be used for later analysis if required. Also, each time Snort is run and then stopped, it displays some details about that session. For instance, the number of packets it received, analyzed, and dropped. Also, the run time of the session, a breakdown of the protocols encountered, and the number of alerts generated are displayed.

4.1.4 Installation of Suricata

Suricata was also installed on the Ubuntu machine. This process consisted of number of steps. Similar to Snort, the first step was to prepare the system for deployment by updating the system. Afterwards, Suricata's dependencies and required packages were installed. The following step was to download Suricata. It can either be downloaded from the source or from the Ubuntu personal package archives. For this project, it was downloaded from Suricata's official website

[37] in order to guarantee that the latest version is installed. Suricata is useless without any rules, hence, the next step was to install the Suricata IDS rule sets. In 2017, Suricata introduced a new rule management tool named Suricata-Update. In order to make sure that all the ruleset is constantly update, this tool was installed. This tool used the Emerging Threats Open ruleset, that includes 27397 rules to detect different intrusions. The final step was to configure Suricata according the system and specify the home network and external network.

Similar to Snort, Suricata has a customisation features that allow users to specify rules to detect particular attacks. Suricata was then tested using the same approach used to test Snort, by sending ping requests from a different machine, while the Suricata host was the target. However, unlike Snort, Suricata had a rule to alert on ping requests without the need to specify one. Another difference is while running Suricata in real-time, it does not print the alerts to the screen as they are coming in. It stores each alert in a log file that can be viewed using a certain command after the traffic is monitored. Figure 13 illustrates a sample results of the testing:



```
04/24/2019-15:35:47.744799 [**] [1:2100366:8] GPL ICMP_INFO PING *NIX [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.185:8 -> 192.168.1.151:0
04/24/2019-15:35:47.744799 [**] [1:2100368:7] GPL ICMP_INFO PING BSDtype [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.185:8 -> 192.168.1.151:0
```

Figure 12. Suricata's Ping Alert

4.2 Rules

The initial direction of the project was to evaluate Snort and Suricata against their default rule sets that were provided using Pulled Pork for Snort and Suricata-update for Suricata. However, after conducting the attacks in all scenarios, it was observed that the majority of attack were not detected by the IDSs, which implies that the rules defined in the default rulesets do not cover such attacks. It was also observed that when the IDSs alert for an attack, the types of alerts were general and not specific to a particular attack. For instance, in the case of a TCP scan, Snort and Suricata would classify the alerts as 'Attempted Information Leak' and 'Potentially Bad Traffic', but not specifically as a TCP Scan.

Therefore, in an attempt to increase the detection rates of the IDSs, specific rules were created for both Snort and Suricata that enabled the IDSs to detect some of the attacks that were not detected previously using the default rule sets. Rules were specifically created for UDP flood, PUSH ACK flood, SYN flood, FIN Scan, XMAS Scan, and TCP Scans. Files were created to include these rules. These files were then added to the configuration files of Snort and Suricata to enable the IDSs to check these rules while analysing. The default rule sets that Snort and

Suricata deploy are extremely similar, hence, the syntax of their rules is also similar. Both IDSs use a simple, lightweight rules description language that is flexible and quite powerful.

A rule is divided into two logical sections, namely the rule header and the rule options. Figure 14 provides an example of a rule. This rule allows the displaying an alert message once a TCP packet is detected.

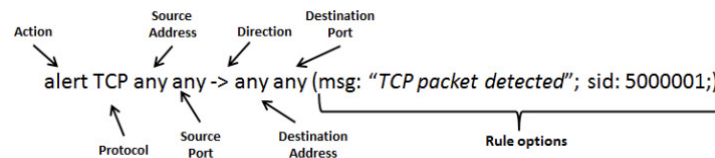


Figure 13. Example Rule [40].

4.2.1 DoS Rules

The following rules allowed detection of **UDP flood** attack traffic:

`alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:" UDP flood attack detected"; threshold: type threshold, track by_dst, count 100, seconds 60; sid: 1000001;).` Where:

- The **msg** keyword contains the message that will be displayed once this attack traffic is detected.
- The **threshold** keyword means that this rule logs every 100th event on this SID during a 60 second interval. So, for instance, if less than 90 events occur in 60 seconds, nothing gets logged. Once an event is logged, a new time period starts for `type=threshold`.
- The **track by_dst** keyword means track by destination IP.
- The **count** keyword means count number of events.
- The **seconds** keyword means time period over which count is accrued.
- The **sid** keyword is used to uniquely identify Snort rules.

The following rule allowed the detecting of **PUSH ACK flood** attack traffic:

`alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" PUSH ACK flood attack detected"; flags: PA; threshold: type threshold, track by_dst, count 100, seconds 60; sid: 1000002;).` Where:

- The **flags** keyword is used to check if the PUSH, ACK flag is set.

The following rule allowed the detecting of **SYN flood** attack traffic:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" SYN Flood attack detected";  
flags: S; threshold: type threshold, track by_dst, count 100, seconds 60; sid: 1000006;).
```

4.2.2 Port Scanning Rules

The following rule allowed the detection of **FIN Scan** traffic:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" SCAN FIN"; flags: F; flow:  
stateless; classtype: attempted-recon; sid: 1000003;). Where:
```

- The **flow** keyword allows rules to only apply to certain directions of the traffic flow, where in this case, it was set to stateless. This means that an alert should be triggered regardless of the state of the stream processor.
- The **classtype** keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class. Snort provides a default set of attack classes that are used by the default set of rules it provides. Defining classifications for rules provides a way to better organize the event data Snort produces. In this case, it was set to attempted-recon, which means that the attack is categorized as an ‘Attempted Information Leak’.

The following rule allowed the detection of **XMAS Scan** traffic:

- alert tcp \$EXTERNAL_NET any -> \$HOME_NET any (msg:" SCAN XMAS"; flags: FPU; flow: stateless; classtype: attempted-recon; sid: 1000004;).

The following rule allowed the detection of **TCP Scan** traffic:

- alert tcp \$EXTERNAL_NET any -> \$HOME_NET any (msg:" SCAN TCP"; flow: stateless; sid: 1000005;).

5. Discussion and Results

This chapter of the report will discuss the results of evaluating Snort and Suricata in three different scenarios. For each scenario, two experiments were conducted. The first one was evaluating the IDSs against their default rulesets. The second experiment was to conduct the evaluation against the default rulesets and with specifying rules for the attacks that were undetected by Snort or Suricata in the first experiment, where the evaluation metrics were accuracy, which was measured by the detection rate, and capability, which was measured by the percentage packets dropped. Snort and Suricata's detection rates for each type of attack are demonstrated below. The results of each attack were documented using Excel, where the number and types of alerts logged, percentage of dropped packets, and whether the attack was a TP or FN was identified.

5.1 First Scenario Results

Experiment 1:

The results of the experiment 1 in the first scenario, where the IDSs were evaluated in a traditional setting against the default ruleset, is shown below.

Snort		
Attack	Detection Rate	Packets Dropped
Connect Scan	100%	0%
FIN Scan	0%	0%
TCP Scan	100%	0%
XMAS Scan	0%	0%
ARP Spoofing	0%	0%
SYN Flood	80%	13%
UDP Flood	0%	0%
PUSH ACK Flood	20%	0%
IP Spoofing	100%	61%

Table 6. Snort Results in Scenario 1, Experiment 1

Table 6 shows that Snort was only able to detect Connect Scans, TCP Scans, SYN Flood, and IP Spoofing attacks using the default ruleset. The SYN Flood attack was conducted 10 times instead of 5 times like the other attacks because its results were not always consistent, therefore, additional tests were conducted for verification. Also, it can be seen that Snort was not able to analyse all the packets during the SYN Flood and IP Spoofing attacks as the percentages of the average dropped packets were 13 and 61 respectively. One reason for this result could relate to the nature of these attacks. As both attacks are considered to be types of DoS attacks, their aims are to flood the target, which could have overwhelmed Snort and resulted in it dropping packets during the analysis. Some of these dropped packets could have had malicious content within them that would usually be logged as an alert, however, as these packets were not analysed, the number of alerts presented to the user decreased. Moreover, the PUSH ACK flood had an anomaly result during one of its tests where it was able to detect the attack by alerting the user with an ‘Detection of a Non-Standard Protocol or Event’ alert. However, during the rest of the tests, it was not able to detect it. The overall detection rate of Snort in this case was 48%.

Suricata		
Attack	Detection Rate	Packets Dropped
Connect Scan	100%	0%
FIN Scan	0%	0%
TCP Scan	100%	0%
XMAS Scan	20%	0%
ARP Spoofing	0%	0%
SYN Flood	90%	0%
UDP Flood	0%	0%
PUSH ACK Flood	0%	0%
IP Spoofing	100%	0%

Table 7. Suricata Results in Scenario 1, Experiment 1.

As illustrated in the table above, Suricata had similar results to Snort when it comes to the types of attacks it was able to detect. However, its ability to handle heavy traffic appeared to be better than Snort as it was able to analyse all the packets in all of the tests conducted. Even though the majority of the XMAS Scans were not detected, during one of the tests, Suricata was able to detect the attack as it provided an alert classified as “Potentially Bad Traffic”. This result was considered an anomaly case as Suricata was not able to detect the attack in any other test. The overall detection rate of Suricata in this case was 50%.

Experiment 2:

In the second experiment, the attacks that were undetected in the experiment 1 were tested again, however, this time, with the specification of rules that attempt to particularly detect them, with the exclusion of the ARP Spoof attack as the specification of a rule to detect it was infeasible. The results for both Suricata and Snort are demonstrated in table 8 below.

Snort		
Attack	Detection Rate	Packets Dropped
FIN Scan	100%	0%
XMAS Scan	100%	0%
UDP Flood	100%	61%
PUSH ACK Flood	100%	0%

Table 8. Snort Results in Scenario 1, Experiment 2

Suricata		
Attack	Detection Rate	Packets Dropped
FIN Scan	100%	0%
XMAS Scan	100%	0%
UDP Flood	100%	0%
PUSH ACK Flood	100%	0%

Table 9. Suricata Results in Scenario 1, Experiment 2

As shown in the tables above, after the specification of rules, the detection rates of attacks in both Snort and Suricata significantly improved when compared with the experiment 1. Also, in this case, the user was alerted with a message that explicitly defined the attack detected, unlike the type of alerts generated in the first experiment. An interesting result was that the average of packets that Snort dropped during the analysis of UDP flood attacks changed from 0% in the first experiment to 61% in this second one. This change could have been a result of the inclusion of a rule to detect this attack in Snort's rules. In this case, Snort had to alert the user and process a significant amount of traffic at the same time, which could have overwhelmed it and caused it to drop packets. The overall detection rate of Snort in this case was 86%.

5.2 Second Scenario Results

Experiment 1:

The results of experiment 1 in the second scenario, where the IDSs were evaluated in IoT against the default ruleset, are shown in tables 10 and 11.

Snort		
Attack	Detection Rate	Packets Dropped
Connect Scan	100%	0%
FIN Scan	0%	0%
TCP Scan	0%	0%
XMAS Scan	0%	0%
ARP Spoofing	0%	0%
SYN Flood	0%	70%
UDP Flood	0%	0%
PUSH ACK Flood	0%	0%
IP Spoofing	100%	82%

Table 10. Snort Results in Scenario 2, Experiment 1.

In the second scenario, Snort was only able to detect Connect Scans and IP Spoofing attacks. This decrease in detection rate could be a result of how the evaluation was setup. For instance,

the reason for why SYN flood attacks was detected in a traditional setting but not with the involvement of the IoT device is because the attacks were conducted from the same host machine of Snort. In this setup, Snort was monitoring the activity of the attacker, who was generating significant amounts of SYN packets against the IoT device. This could have affected Snort's capability to handle traffic due to the heavy traffic it was encountering, resulting in packets being dropped. The average of dropped packets for this type of attack increased from 13% in the first scenario to 70% in this scenario, which implies that Snort was not able to monitor the majority of packets that could have indicated the attack; thus, it was not able to alert an attack. The overall detection rate of Snort in this case was 20%.

Suricata		
Attack	Detection Rate	Packets Dropped
Full Scan	100%	0%
FIN Scan	0%	0%
TCP Scan	100%	0%
XMAS Scan	0%	0%
ARP Spoofing	0%	0%
SYN Flood	0%	0%
UDP Flood	0%	0%
PUSH ACK Flood	0%	0%
IP Spoofing	100%	0%

Table 11. Suricata Results in Scenario 2, Experiment 1

As demonstrated in table 11, Suricata performed better than Snort in this setting as it was able to detect three attacks instead of two types of attacks, which were full scans, TCP scans, and IP spoofing attacks. However, the majority of the attacks were still undetected. Also, Suricata was able to analyse all packets in all tests with none of the packets being dropped. However, unlike its detection in the first scenario, Suricata was not able to detect SYN flood attacks in this setting. The overall detection rate of Suricata in this case was 30%.

Experiment 2:

Snort		
Attack	Detection Rate	Packets Dropped
FIN Scan	100%	0%
TCP Scan	100%	0%
XMAS Scan	100%	0%
SYN Flood	100%	58%
UDP Flood	100%	46%
PUSH ACK Flood	100%	0%

Table 12. Snort Results in Scenario 2, Experiment 2

Suricata		
Attack	Detection Rate	Packets Dropped
FIN Scan	100%	0%
XMAS Scan	100%	0%
SYN Flood	100%	0%
UDP Flood	100%	0%
PUSH ACK Flood	100%	0%

Table 13. Suricata Results in Scenario 2, Experiment 2

It can be seen from table 12 and table 13 above that detection rates have improved dramatically when attack specific rules were identified in the second scenario. In addition to the rules specified in the previous scenario, additional rules that would detect TCP Scans and SYN floods were created, as they were not detected by the default rules when applied on the IoT device. The average of dropped packets by Snort when it is faced with heavy traffic resulting from DoS attacks was still high, which implies limitations in its capability. This was demonstrated by an average of 58% of packets dropped in the case of a SYN floods and an average of 46% for UDP floods. On the other hand, Suricata's capability to handle packets is still consistent in this scenario as the average of packets dropped in all tests remains 0%. The overall detection rate of both Suricata and Snort in this case was 90%

5.3 Third Scenario Results

In this third scenario, the evaluation metrics differed from the previous scenarios. As Snort and Suricata were run offline, evaluating the capability became irrelevant as the IDSs were not acquiring packets in real-time, thus, packets would not be dropped. The IDSs would not show the results of each test unless all the packets are analysed. Also, each attack was tested once as the content of the PCAP files, which contained the attack traffic combined with real background traffic, would not change in each iteration like in the previous scenarios. Therefore, the accuracy was evaluated by whether an attack was detected or not when its PCAP file was run with Snort and Suricata. An overall detection rate was identified for each scenario, which was represented by the ratio of attack types it was able to detect and the total amount of attack types tested.

Experiment 1:

Snort	
Attack	Detected
Full Scan	Yes
FIN Scan	No
TCP Scan	Yes
XMAS Scan	No
ARP Spoofing	No
SYN Flood	No
UDP Flood	No
PUSH ACK Flood	No
IP Spoofing	Yes

Table 14. Snort Results in Scenario 3, Experiment 1

Suricata	
Attack	Detected
Full Scan	Yes
FIN Scan	No
TCP Scan	Yes
XMAS Scan	No
ARP Spoofing	No
SYN Flood	No
UDP Flood	No
PUSH ACK Flood	No
IP Spoofing	Yes

Table 15. Suricata Results in Scenario 3, Experiment 1

The accuracy of Snort and Suricata is illustrated in table 14 and table 15. In this scenario, both IDSs achieved similar accuracy as they were able to detect three out of the nine types of attacks. This included the full scan, TCP scan, and IP spoofing attacks. The overall detection rate of Snort and Suricata in this case was 33%.

Experiment 2:

Snort	
Attack	Detected
FIN Scan	Yes
XMAS Scan	Yes
SYN Flood	Yes
UDP Flood	Yes
PUSH ACK Flood	Yes

Table 16. Snort Results in Scenario 3, Experiment 2

Suricata	
Attack	Detected
FIN Scan	Yes
XMAS Scan	Yes
SYN Flood	Yes
UDP Flood	Yes
PUSH ACK Flood	Yes

Table 17. Suricata Results in Scenario 3, Experiment 2

It can be seen from table 16 and table 17 above that both Snort and Suricata were able to detect the remaining attacks when attack specific rules were specified. Their overall detection increased to 90% in this experiment versus the 33% in experiment 1.

5.4 Summary of Results

Based on the results illustrated in the previous section, it can be realised that the performance of Snort and Suricata in IoT and a traditional setting in terms of accuracy was high when specific rules to detect the attacks were specified. In contrast, their detection performance was considerably poor using only the default rulesets. Also, using the default rules, the alerts the user would receive were general and not attack specific. However, by using attack specific rules, the detection was improved as the user was receiving alerts that explicitly indicated the type of attack detected.

When it comes to the processing of packets, Suricata performed extremely well in all scenarios. The former could be credited to its multithreaded capturing technique. As mentioned in the background, this technique allows for multiple-packet captures queues for each worker process, which distributes the workload across multiple CPU cores available. In contrast, Snort was unable to analyse all packets in some cases, which resulted in it dropping packets. This was particularly in the case of DOS attacks as this type of attack aims to flood the system with heavy traffic. In order to further demonstrate the effect of heavy traffic on Snort in terms of its capability to analyse packets, an additional experiment was conducted. In this experiment, different numbers of packets in each iteration were sent to flood the target, where the number of packets sent would increase in each iteration. Figure 15 below demonstrates the results of the experiment.

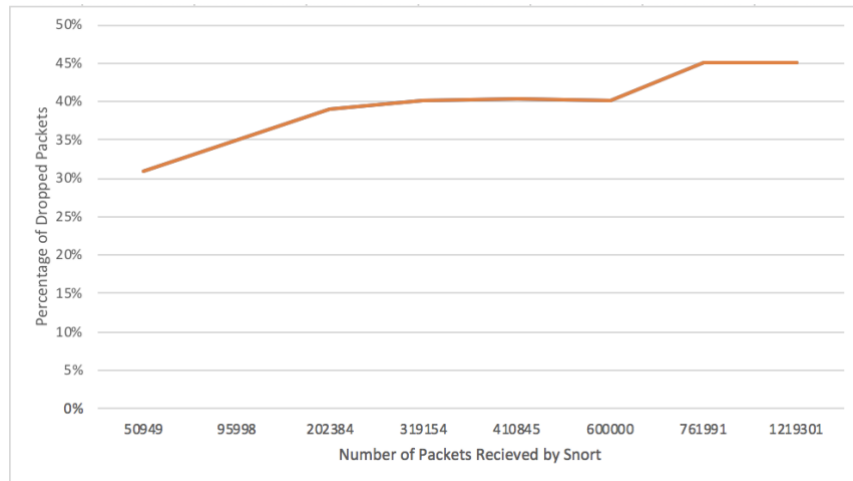


Figure 14. Results of Snort Capability with Heavy Traffic

It can be seen from this figure that with the increase in the number of packets that Snort receives, the more it tends to drop packets. This could indicate that is not suitable for deployment in heavy traffic networks.

Based on these results, it can be concluded that the traditional IDSs Snort and Suricata are suitable in IoT environments in terms of detection of attacks as they work particularly well when attack specific rules are specified. When it comes to their capability to analyse traffic, it was shown that Suricata is able to fully analyse heavy traffic, whereas Snort would occasionally drop packets in the case of heavy traffic. However, the main concern with deploying these IDSs in IoT is the placement of the IDS. As these IDSs are not installed on the IoT device itself, alternative methods to obtain the traffic of such devices must be considered in order to provide the IDSs with the traffic of the device they should monitor. This could indicate that they might not be suitable for deployment in IoT by the typical user due to the complexity of the task.

6. Final Thoughts

6.1 Conclusion

The number of IoT devices is increasing and becoming more common, and with new functionality comes new security threats to an area where security has not been the priority. The utilisation of an intrusion detection system could assist with the detection of attacks targeting these devices. The main aim of this project was to evaluate Intrusion Detection Systems that are created for traditional environments in an Internet of Things environment. Two open source and commonly used IDSs were evaluated, Snort and Suricata. To achieve this, several types of attacks were generated against an IoT device, where the traffic was analysed by the IDSs in order to observe their behaviour against such attacks. These attacks consisted of three categories, Port Scanning, Man in the Middle, and Denial of Service attacks. The two main evaluation metrics were accuracy, which was represented by the detection rate, and capability, which was represented by the average of dropped packets. The evaluation was conducted in three scenarios. The first scenario was to evaluate the IDSs in real-time in a traditional setting, the second scenario to evaluate in real-time in an IoT setting, and third scenario to evaluate the IDSs in their offline modes in an IoT setting with the inclusion of real background traffic. In each scenario, two experiments were conducted, where the IDSs were once evaluated using their default rulesets and once with the inclusion of specified rules to detect the attacks. The main finding of this report was that traditional IDSs' accuracy when applied to IoT was low with the default rulesets, as a maximum of three out of nine attack types were detected in this case. However, the accuracy of the IDSs improved dramatically when attack specific rules were defined and used. It was also found that Snort's capability to handle heavy traffic was low. This was highlighted by the average of packets dropped by Snort when it was faced with Denial of Service type of attacks, as this attack aims to overwhelm the system with heavy traffic. On the other hand, it was found that Suricata's capability was high when encountered with such traffic as it had an average of 0% of dropped packets in all cases. This report has provided an insight into the suitability of traditional IDSs in IoT, where it was highlighted that the main concern was obtaining the required traffic for monitoring,

6.2 Future Work

Due to the limited time allowed for this project, there are several aspects of the evaluation that could potentially be improved in the future.

One aspect is the automation of the evaluation process. In the future, a framework could be created that would automatically generate attack scenarios targeting the IDS host machine, where the alerts produced by the IDS would be collected and analysed automatically. This could be done by identifying the number of detected attacks, True Positives, and the number of undetected attacks, False Negatives to calculate the detection rate. Reports can then be generated that would display the average of dropped packets and detection rate of the IDS for each attack.

The next aspect that could be further explored is the number and types of IoT devices to be included in the evaluation. Currently, only one device provides the IoT element in the evaluation. However, in future work, the IDSs could be evaluated in an environment where multiple IoT devices of different types are connected and are communicating with one another.

Also, in the future, the IDSs could be evaluated against additional types of attacks that are different than the ones generated in this evaluation in order to further observe the accuracy of these IDSs. For example, Snort and Suricata could be evaluated against attacks that are specific to IoT, such as the Mirai malware.

Additionally, as the IDSs in this evaluation setting were not placed on the IoT device itself, some evaluation metrics were not applicable. In future work, installing the IDSs on an IoT device, for instance, a Raspberry Pi device, could be attempted. The former would allow the evaluation of the performance of the IDSs in terms of CPU utilization and memory consumption, which would provide a more inclusive evaluation of the IDSs.

6.3 Reflection on Learning

While working on this project, I gained a number of technical skills and increased my knowledge in various areas. When I started the project, my knowledge in Internet of Things and the deployment of intrusion detection systems was very little. However, by the end of this project, I achieved studying and understanding these concepts, and became very familiar with two different IDSs, Snort and Suricata. This technical knowledge I acquired will be helpful in my future career as after graduation, I will be employed in a department that focuses on the security of the company's systems. For instance, I can use the knowledge I gained about generating various attacks and the use of IDSs to perform penetration testing on any new systems my company creates. Also, one of the main skills I gained is the use of command line in all aspects of the project. In my previous assignments and work, I would always prefer to use a user interface to program and develop with. However, in this project, all the tools used to perform the attacks and the process of installing and running both Snort and Suricata required the use of command line, therefore, I got to develop this important skill and learn many new commands.

On the other hand, I was faced with a number of obstacles and challenges. The main challenge faced was getting Snort and Suricata to see the traffic of the IoT device, as it was infeasible to install them on the device. However, I managed to find alternative methods in order to overcome this challenge. The lack of documentation when it comes to Snort and Suricata was also one of the main obstacles. This resulted in difficulties and spending significant time attempting to install and configure these IDSs. Nevertheless, I was able to eventually install and deploy them in a successful manner.

Moreover, valuable lessons were learned while working on the project. The main lesson was to not waste time on a particular issue and try to find alternatives. At the beginning of the project, significant time was spent on trying to install Bro IDS, which delayed the progress of this project, where finally Suricata was evaluated instead to overcome this shortcoming. Another lesson learned was the importance of having proper time management. As this is the longest-term project of this nature I have ever undertaken, the possibility of procrastination was high, however, I managed to organise my time and minimise any distractions in order to produce quality work and meet the deadline.

Additionally, throughout the project, I called upon knowledge from my studies at university. I used my understanding of different attacks that was taught in the security module to generate some of the attacks, such as the Man in the Middle attack. I also benefited from the knowledge that I gained from the Communication Networks and Pervasive Computing module. This knowledge provided me with a basis for understanding the various aspects of networks and some monitoring tools used.

References

- [1] AM, R. 2017. Intrusion Detection System Techniques and Tools: A Survey. *Scholars Journal of Engineering and Technology* 5(3), pp. 122-133.
- [2] Alserhani, F. et al. ND. Snort Performance Evaluation [Online]. Available at: https://pdfs.semanticscholar.org/d441/b3833899e3952422722fcf06acda3271981c.pdf?_ga=2.130461255.916643669.1556108268-1304794693.1556108268 [Accessed: 26 April 2019].
- [3] Antonakakis, M. et al. 2017. Understanding the Mirai Botnet [Online]. Available at: <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf> [Accessed: 7 May 2019].
- [4] AlSukkar, G. et al. 2016. Address Resolution Protocol (ARP): Spoofing Attack and Proposed Defence. *Communications and Network*, 8, pp. 118-130.
- [5] Baccelli, E. et al. 2012. RIOT: One OS to Rule Them All in the IoT [Online]. Available at: https://www.researchgate.net/publication/259688444_RIOT_One_OS_to_Rule_Them_All_in_the_IoT [Accessed: 20 March 2019].
- [6] Belkasmi, M. et al. 2017. Implementation and Performance Evaluation of Intrusion Detection and Response System. *Proceeding of Engineering and Technology*, 19, pp. 12-16.
- [7] Bhosale, D. and Mane, V. 2015. Comparative study and analysis of network intrusion detection tools. *International Conference on Applied and Theoretical Computing and Communication Technology*, Davangere, 2015, pp. 312-315.
- [8] Bhuvaneswari, V. and Porkodi, R. 2014. The Internet of Things (IoT) Applications and Communication Enabling Technology Standards: An Overview [Online]. Available at: https://www.researchgate.net/publication/286573266_The_Internet_of_Things_IoT_A

pplications_and_Communication_Enabling_Technology_Standards_An_Overview
[Accessed: 5 Feb 2019].

- [9] Burhan, M., Rehman, R., Khan, B. and Kim, B. 2018. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. *Sensors*, 18(9), p.2796.
- [10] Dsniff. 2019. Abstract [Online]. Available at: <https://monkey.org/~dugsong/dsniff/> [Accessed 29 April 2019].
- [11] ElHajj, W. et al. 2014. Real traffic logs creation for testing intrusion detection systems. *Wireless Communications and Mobile Computing*, 15(14), pp.1851-1864.
- [12] Elrawy, M. et al. 2018. Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing: Advances, Systems and Applications* 7(1).
- [13] Ettercap. 2019. About [Online]. Available at: <https://www.ettercap-project.org/about.html> [Accessed 29 April 2019].
- [14] EY Global. 2019. [online] Available at:
[https://www.ey.com/Publication/vwLUAssets/ey-m-e-internet-of-things/\\$FILE/ey-m-e-internet-of-things.pdf](https://www.ey.com/Publication/vwLUAssets/ey-m-e-internet-of-things/$FILE/ey-m-e-internet-of-things.pdf) [Accessed: 31 Jan. 2019].
- [15] Hassan, Z. et al. 2015. Internet of Things (IoT): Definitions, Challenges and Recent Research Directions. *International Journal of Computer Applications*, 128(1), pp.37-47.
- [16] Hping. 2019. Documentation [Online]. Available at:
<http://www.hping.org/documentation.php> [Accessed 21 April 2019].
- [17] Imperva. 2019. DDoS Attack Types & Mitigation Methods [Online] Available at:
<https://www.imperva.com/learn/application-security/ddos-attacks/> [Accessed: 13 April 2019].

- [18] IoT Agenda. 2019. What is a man-in-the-middle attack? [Online]. Available at: <https://internetofthingsagenda.techtarget.com/definition/man-in-the-middle-attack-MitM> [Accessed: 12 April 2019].
- [19] Jun, C. and Chi, C. 2014. Design of Complex Event-Processing IDS in Internet of Things [Online] Available at https://www.researchgate.net/publication/261959979_Design_of_Complex_Event-Processing_IDS_in_Internet_of_Things [Accessed 22 March 2019].
- [20] Kadam, P. and Deshmukh, M. 2014. Various Approaches for Intrusion Detection System: An Overview. International Journal of Innovative Research in Computer and Communication Engineering 2(11), pp. 2320-9798.
- [21] Kasinathan, P. et al. 2013. Denial-of-Service detection in 6LoWPAN based Internet of Things [Online]. Available at: https://www.researchgate.net/publication/261128514_Denial-of-Service_detection_in_6LoWPAN_based_Internet_of_Things [Accessed 24 April 2019].
- [22] Khan, R. et al. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges [Online]. Available at: https://www.researchgate.net/publication/261311447_Future_Internet_The_Internet_of_Things_Architecture_Possible_Applications_and_Key_Challenges [Accessed: 15 April 2019].
- [23] Kumar, G. 2014. Evaluation Metrics for Intrusion Detection Systems - A Study. International Journal of Computer Science and Mobile Applications, 2(11), pp 11-17.
- [24] Leyden, J. 2019. Samsung smart fridge leaves Gmail logins open to attack [Online] Available at: https://www.theregister.co.uk/2015/08/24/smart_fridge_security_fubar/ [Accessed: 15 April 2019].
- [25] Lobo, S. 2018. The 10 most common types of DoS Attacks [Online]. Available at: <https://hub.packtpub.com/10-types-dos-attacks-you-need-to-know/> [Accessed 13 April 2019].

- [26] Massicotte, F. et al. 2006. Automatic Evaluation of Intrusion Detection Systems [Online] Available at: <https://cglab.ca/~mathieu/ACSAC06.pdf> [Accessed: 7 March 2019].
- [27] Metasploit. 2019. Metasploit Basics [Online]. Available at: <https://metasploit.help.rapid7.com/docs/metasploit-basics> [Accessed 10 April 2019].
- [28] Milenkoski, A. Evaluation of Intrusion Detection Systems in Virtualized Environments [Online]. Available at: https://research.spec.org/fileadmin/user_upload/dissertationaward/Aleksandar_Milenkoski_Dissertation.pdf [Accessed: 15 April 2019].
- [29] Nalavade, K. and Meshram, B. 2011. Comparative Study of IDS and IPS. International Journal of Agriculture Sciences 2(11), pp 1- 4.
- [30] Nmap. 2019. Inroduction [Online]. Available at: <https://nmap.org/> [Accessed 22 April 2019].
- [31] Pihelgas, M. 2012. A Comparative Analysis of Opensource Intrusion Detection Systems. PhD Thesis, Tallinn University of Technology.
- [32] Sethi, P. and Sarangi, S. 2017. Internet of Things: Architectures, Protocols, and Applications. Journal of Electrical and Computer Engineering, 2017, pp.1-25.
- [33] Sforzin, A. et al. RPiDS: Raspberry Pi IDS A Fruitful Intrusion Detection System for IoT [Online]. Available at: <https://ants.inf.um.es/~felixgm/pub/conferences/16/GomezMarmol-ATC16.pdf> [Accessed 19 March 2019].
- [34] Shah, S. and Issac, B. 2018. Performance comparison of intrusion detection systems and application of machine learning to Snort system. Future Generation Computer Systems, 80, pp.157- 170.

- [35] Shimeall, T. and Spring, J. 2014. Introduction to information security. Amsterdam: Elsevier.
- [36] Snort - Network Intrusion Detection & Prevention System. 2019. Available at: <https://www.snort.org/> [Accessed: 10 Feb 2019].
- [37] Suricata. 2019. Available at: <https://suricata-ids.org/> [Accessed: 10 May 2019].
- [38] Swinhoe, D. (2019). What is a man-in-the-middle attack? How MitM attacks work and how to prevent them [Online]. Available at: <https://www.csoonline.com/article/3340117/what-is-a-man-in-the-middle-attack-how-mitm-attacks-work-and-how-to-prevent-them.html> [Accessed: 12 April 2019].
- [39] Tayyebi, Y. and Bhilare, D. 2018. A Comparative Study of Open Source Network Based Intrusion Detection Systems. International Journal of Computer Science and Information Technologies 9(2), pp 23-26.
- [40] Trabelsi, Z. and Alketbi, L. 2013. Using network packet generators and snort rules for teaching denial of service attacks [Online]. Available at: https://www.researchgate.net/publication/262345832_Using_network_packet_generators_and_snort_rules_for_teaching_denial_of_service_attacks [Accessed 28 April 2019].
- [41] Tripathi, N. and Mehtre, B. 2013. DoS and DDoS Attacks: Impact, Analysis and Countermeasures [Online]. Available at: https://www.researchgate.net/publication/259941506_DoS_and_DDoS_Attacks_Impact_Analysis_and_Countermeasures [Accessed: 13 April 2019].
- [42] Wireshark. 2019. About Wireshark [Online]. Available at: <https://www.wireshark.org/index.html#aboutWS> [Accessed 14 April 2019].
- [43] Zarpelaoa, B. et al. 2017. A survey of intrusion detection in Internet of Things [online]. Available at: <http://www.download-paper.com/wp-content/uploads/2017/11/2017-elsevier-A-survey-of-intrusion-detection-in-Internet-of-Things.pdf> [Accessed: 1 Feb. 2019].

[44] Zeek Network Security Monitor. 2019. Available at: <https://www.zeek.org/> [Accessed: 2 April 2019].