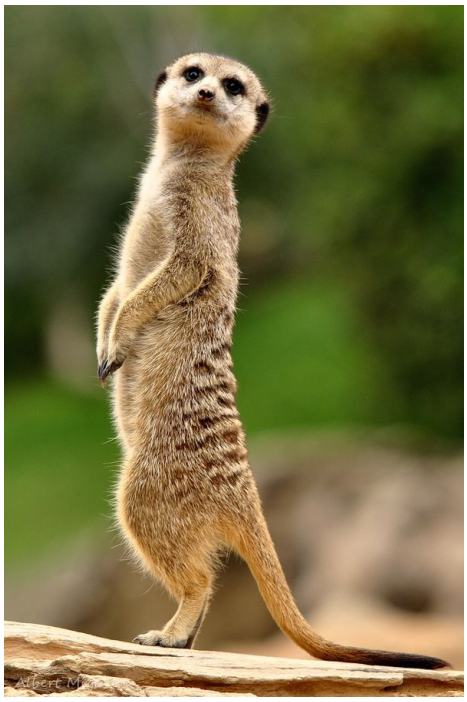
COMS4200/ 7200 Project Proposal
Exploring Suricata



https://images.app.goo.gl/9NEAWv7nJdJCSkZQA

List of Team Members

Name	**Email**	**Course Code**
Ebrahim Awad S Alharbi	s4564841@student.uq.edu.au	COMS7200
William Cumines	s4391890@student.uq.edu.au	COMS4200
Lachlan Drury	s4478191@student.uq.edu.au	COMS4200
Harry Huang	s4530915@student.uq.edu.au	COMS4200
Kausthubram Rajesh	s4479071@student.uq.edu.au	COMS4200

Table of Contents

1. Abstract

As computer networks get more complex, attacks into these networks are also becoming more prominent. Hence, it is important to have systems to detect and prevent these attacks. This is where IDS and IPS applications such as Suricata come into play.

Through background research on the relevant topics and related works such as Network Security and Software Defined Network concepts; our team has been able to narrow down our scope and methodology for the project.

The overall aim of the project is to simulate DOS attacks within an SDN environment and test the response of Suricata for detecting and preventing these attacks with changes to various experimental variables.

For this project, our team has created a detailed plan listing major timeline tasks and milestones. We are tracking well in terms of progress, where the team has been able to apply background research into performing simple experiments to test the basic abilities of Suricata. Following the Project Proposal, we plan on running more complex experiments and simulations of attacks to test the effectiveness of Suricata. We plan to report our findings to help inspire the use of Suricata or other similar applications in university and enterprise networks.

2. Introduction

2.1 Overview

These days our daily lives depend on several computer network applications, so they are attractive targets for attackers through malicious activities. As a result, various techniques were developed to ensure the security of computer systems and networks. One of the essential techniques is Network Intrusion Detection Systems (NIDS).

2.2 Network Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) detect and even prevent intrusion into computer systems or networks by providing information on susceptible activities [1]. They have two types: host-based systems which collect information about a particular system or host (computer), network-based systems which collect information from its whole network rather than a separate host [2]. NIDS are classified into two major categories: anomaly-based (behaviour-based) and signature-based (knowledge-based) [1]. The anomaly-based NIDS establish a normal profile and generate alerts when different behaviour was observed. Signature-based NIDS detect threats and generate alerts through using predefined attack signatures.

There are a variety of open and commercial NIDS such as Snort and Suricata. Since this project will focus on exploring Suricata, the following section introduces it.

2.3 Suricata

Monitoring network security is crucial because it is the first step to detect and prevent several threats. For example, when analysts notice suspicious activity on a network, they will start a further investigation. Therefore, the Intrusion Detection System(IDS), Intrusion Prevention System(IPS), and Network Security Monitoring (NSM) technologies play an essential role in protecting the security of the networks.

Suricata provides all the above capabilities, so it is crucial to protect the networks against the threats and attacks.It inspects the network through extensive rules and signature language to detect the threats [3]. In addition, it supports the Lua programming language. Also, Suricata has high performance because it is multi-threaded.

Suricata can detect and protect several types of attacks. It can detect many known attacks. One of them is SQL injection, which is a dangerous attack for exploiting web applications [4]. In addition, it can detect Local File Inclusion (LFI), which the attacker injects path to include files on the web server [4]. For example, 8.3 records of Freepik and Flaticon users were stolen through a SQL injection attack in 2020 [5].

In addition, Suricata can detect several evasion techniques[4]. One of them is payload mutation through encoding payload. Another one is packet splitting, where the attacker evades IDS through splitting IP datagram into overlapping fragments or TCP stream into overlapping segments [4]. Moreover, it can detect many types of Denial Of Service (DOS) attacks such as Hping syn flood and FTP brute force [4]. For example, attackers targeted a UK college with DoS attacks on the results day [6].

Moreover, Suricata can detect many malware such as trick bot, which redirects the user on a malicious page, and word documents push Hermes ransomware, which the attacker email the victim with word document protected with a password and asks him/her to enable the macro, so ransomware will be downloaded and encrypts all the files. In addition, Suricata can detect shellcode mutations like shellcode encoded with base64 [4].

Suricata mainly can be used as host-based IDS to monitor the traffic of a single host. Also, it can be used as passive IDS, which alerts the network administrators, or active inline IDS, IPS, which prevents the threat and alerts the administrators.

2.4 Project Goal

The goal of this project is to identify the capabilities of Suricata, in order to make informed decisions in the future; with the relevant context of applying our findings to university and enterprise networks. This will be done empirically through a series of experiments to see how Suricata can be used to stop a series of different penetration attacks.

3. Background

3.1 Network Security Overview

A firewall is a network security system that monitors and controls inbound and outbound traffic based on given rules. The main goals for a firewall are: enforcing security policies; ensure all traffic from internal networks to the Internet, and vice versa, must pass through the firewall; only allow traffic authorised by policy to pass. The types of rules applied to traffic to be allowed to pass through includes: address ranges, protocols, applications and content types. Firewalls can't protect traffic that does not cross such as internal traffic, it will also fail when misconfigured [7].

An Intrusion Detection System (IDS) is a physical device, or software application that monitors a network system for malicious activity or policy violations. Primary assumptions of using an IDS is that the system activities are observable with normal and intrusive activities having distinct evidence. IDS use two main types of detection techniques, Anomaly and Signature detection. Anomaly Detection involves the collection of data relating to the behaviour of legitimate users over a period of time. Current observed behaviour is analysed to determine whether this behavior is that of a legitimate user or that of an intruder. Signature (Misuse) Detection uses a set of known malicious data patterns or attack rules that are compared with current behavior. This technique can only identify known attacks for which it has patterns or rules [6].

Intrusion Prevention System (IPS) is an extension of an IDS that includes the capability if attempting to block or prevent detected malicious activity. IPS can be host-based, network-based, or distributed/ hybrid. IPS uses Anomaly Detection to identify behaviour that is not that of legitimate users, or Signature Detection to identify known malicious behaviour can block traffic like a firewall does; it makes use of the algorithms developed for IDSs to determine when to do so [7].

3.2 SDN breakdown

Software defined networking (SDN) is a networking architecture that aims to redefine how networks are built, to counteract the ever-growing complexity and rigidity of traditional networks [8]. Traditional networks work using a distributed algorithm, executed on every router within the network. Meaning, each router is responsible for both; forwarding all traffic that passes through it to the correct output port, and creating and managing the routing table, from which the router can query for the information to correctly forward traffic to the correct output port. The forwarding of packets to output ports is known as the Data Plane, while the creation and management of the routing table is known as the Control Plane. The primary concept behind SDN is to decouple the data and control planes, placing the control plane in a logically centralized infrastructure. Figure 1 shows this transition from a traditional networking infrastructure to a SDN networking infrastructure.

SDN decouples the data and control planes first by placing an abstraction over the data plane. Currently, the main technology for this is OpenFlow, which is explored in more detail in section 3.3 below. In brief, OpenFlow acts as the interface between the data plane and the control plane, abstracting the functionality of the data plane, so that the control plane is shielded from the vagaries and specifics of the distributed nature of the data plane. The primary advantage of this is that making changes and creating new technologies and innovations within the control plane no longer requires a detailed understanding of what happens within the data plane, as well as removing the potential need to manually configure every router and switch on the network whenever a change to that network is rolled out.

SDN opens the door to a flood of innovation in an area that has been relatively stagnant over the past several years. Some of the key advantages an SDN architecture provides are: enabling complex automation to assist with management tasks that were previously impossible to automate due to the distributed nature of the control plane, access to a multi-vendor network as previously vendor-specific hardware made this too complex to be worthwhile, increases visibility and control over the network allowing the guarantee of quality of service, security, traffic engineering, and access control being enforced consistently between multiple wireless and wired network infrastructures [8].

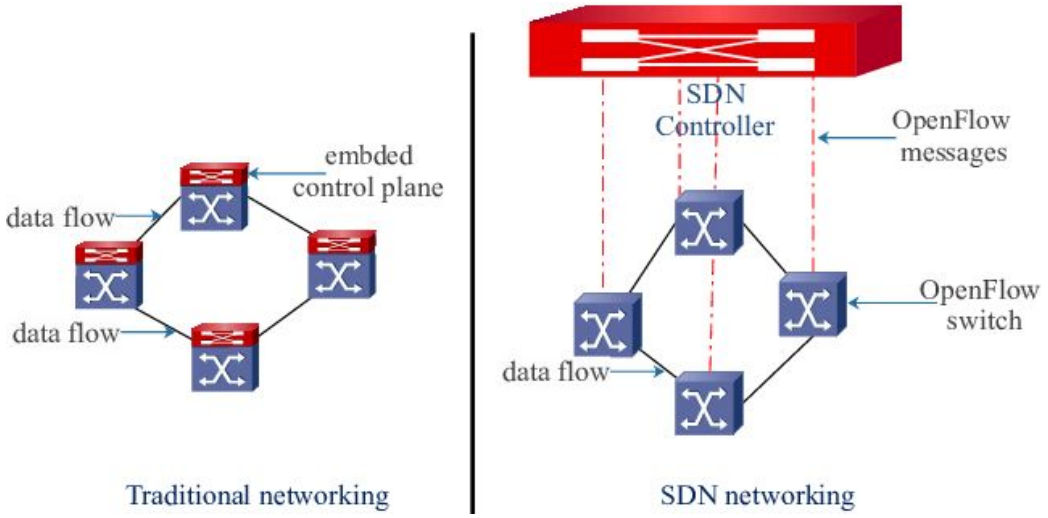


Figure 1. Traditional networking infrastructure compared to SDN networking infrastructure [9, fig. 1].

3.3 Openflow breakdown

Openflow is a standard protocol for communication in SDN networks between switches and controllers. It is used for communication between the control layer and data layer. These layers can be briefly summarised as the control layer being the layer which serves to make the decisions on where to route the packet, whereas the data plane is a series of switches which actually do the forwarding. In standard networks, this is done in the switch and router itself. This is really inefficient as each routing protocol needs to be implemented on top of every company's custom operating system and effectively leads to duplication of the effort. A core concept in software defined networks is the fact there is a separation of the control and data plane with the switches being 'dumb' devices that communicate with controllers which manage the Control plane of the entire network. However, this means that there needs to be a common language that is reliable between these switches and controllers. OpenFlow is this language.

OpenFlow allows you to directly tell switches different actions based on the packets that they have received in a reliable manner. This reliability stems from the fact that it is written on top of TCP which means that efforts will be made by both sides to ensure the otherside has received their query.

The SDN and OpenFlow paradigm has many key advantages over traditional routing frameworks as it allows for central control over switches regardless of which company had manufactured them, more granular control and better user experience. This granular control will be important in the experiments that we intend to run and would not be as simple in traditional networking architectures [10].

3.4 Mininet breakdown

Mininet is a network emulator for several purposes, such as testing, research, and learning [11]. It creates virtual hosts, switches, controllers, and links and supports OpenFlow. It can run 4096 virtual hosts and switches. Its main advantage is providing a cheap and straightforward network testbed, so there is no need to deal with the physical network.

3.5 Suricata breakdown

Suricata is a free and open-source network threat detection engine made by the Open Information Security Foundation (OISF). Suricata is capable of real time Intrusion Detection (IDS), inline Intrusion Prevention (IPS), Network Security Monitoring (NSM) and offline packet capturing. On a high level, Suricata inspects the network traffic using an extensive set of rules and signature language. Suricata supports several standard formats, such as JSON and YAML [1].

The first version of Suricata was released in 2010 [2]. Although Suricata code is original, its developers borrowed from the Snort, which is open source NIDS and was released in 1998, architecture [2]. Open Information Security Foundation (OISF) claims the purpose of Suricata is to bring new ideas and technologies to the field not to replace the existing tools [12]. Although Suricata uses similar rules as Snort, it brings new technologies and ideas such as multi-threading to Intrusion Detection Systems [12]. Therefore, it analyzes the network traffic concurrently, so it achieves high scalability to detect network threats [13].

Suricata gets one packet at a time from the network system of the nfnetlink_queue from the kernel. These packets are then pre-processed, and passed to the detection engine. Suricata works with rules and can either accept or drop packets. Suricata IPS introduces the actions of 'drop', 'sdrop' (silent drop) and 'reject' for rejecting packets.

3.6 DOS Attacks and DOS Protection Techniques

A denial of service attack (DOS Attack) is a network based attack intended to negatively affect legitimate users' availability to a system by saturating the system’s resources. There are two primary methods of a DOS attack, one being through system vulnerabilities, which by malformed network packets utilize such vulnerabilities, “causing excessive memory consumption, extra CPU processing, system reboot, or general system slowing” [14, p. 2]. On the other hand there are flooding attacks, which aim to overload a network or system by sending large amounts of network traffic to the intended victim. This attack can be inflicted by a range of protocols, namely, ICMP, UDP, and TCP [15]. Additionally this can be extended to a Distributed DOS (DDOS) where multiple hosts coordinate together to launch the attack [14].

The detection of a DOS based attack requires the distinction between a malicious and a legitimate packet to ensure no service interruption is made to any legitimate user. As such most detection methods define an attack as an “abnormal and noticeable deviation of some statistic” when monitoring a real-time network workload. The discerning difference in methods being the statistical metric used. One such approach is to create activity profiles by correlating network traffic by inspecting packet headers for similarities such as (address, port, protocol). From this flows are defined as the packet rate for each profile; in which an increase in a flow or a cluster of similar flows could indicate an attack [14].

4. Related Works

4.1 Use case in American telco AT&T

In any organisation, especially telecommunication companies like AT&T, it is very important to implement specific solutions to detect security attacks and threats. In AT&T, a product called USM (Unified Security Management Platform) Anywhere is used. USM Anywhere comes with a network IDS and the software used is open source Suricata [16].

Organisations like AT&T typically use solutions like Host Intrusion Detection Systems (HIDS) and Network Intrusion Detection Systems (NIDS). In NIDS, the two main approaches are signature and anomaly based detection. Suricata is a signature-based IDS and once it is properly configured, it is capable of doing real-time traffic inspection in order to trigger alarms when suspicious activity is detected in the network environment [16].

Suricata is capable of running multiple threads. If an organisation has hardware with multiple CPUs/ cores, this tool can be configured to distribute the workload on several processes at the same time. From AT&T's experimental experience, multi-threading is an excellent configuration to improve Suricata's performance. Suricata has four thread modules: packet acquisition which is responsible for reading packets from the network; decode and stream application layer which decodes the packets and inspects the application; detection which compares signatures and can be run in multiple threads; outputs where all the alarms are processed. Hence, the use of Suricata with multi-threading can be applied to this project, where the application can be configured to test effectiveness by seeing if any false positives/ negatives are wrongly picked up.

Suricata also offers run modes with the names of singlem, works and autofp for its modes. It runs on a Lego-Style approach with threads, thread-modules, and queues working together. Thread-modules are specific thread functionalities, like decode or detect. A packet can be processed by more than one thread and queues are responsible for passing the packet from one thread to another. When those three elements combined work together in packet processing, they become a runmode [16].

Suricata can run in most operating systems namely; Linux, Mac OS, and Windows, with no specific hardware configuration to run it; hence it is very useful for the cybersecurity of big organisations like AT&T. Furthermore, this will be useful for the project where our main experimental machine is intended to be a Linux Virtual Machine. This will allow the simulations of various DOS attacks on simplified versions of an enterprise network topology and using Suricata for detection and prevention.

4.2 Suricata vs Snort

Researchers test and compare Suricata and Snort in terms of detection of known attack, malware and evasion techniques [4]. They collected their general functionalities of the literatures, see Table 1 below.

Functionality	**Suricata**	**SNORT**
Rule	Using rules from emerging threat and VRT, Lua script, possibility of using pulled pork for rule management	Using rules from VRT and doesn't support all rule from emerging threat possibility of using pulledpork for rules management
RAM usage	High	Medium
Dropped Packets	Medium	High in the large networks
Multi CPU	Suricata use multiple CPU	Single CORE processor
IPS functionality	Intrusion Prevention with NFQUEUE	SNORT Inline supports new rule keywords to allow traffic matching a rule to be dropped, thus turning SNORT into an Intrusion Prevention System (IPS)
IPV6 Support	Yes	Yes
Installation and configuration	Easy to install and configure	Complete documentation in www.SNORT.com
Compatibility with OSs.	Windows, UNIX, macOS	Portable (Linux, Windows, macOS X, Solaris, BSD, IRIX, Tru64, HP-UX)

Table 1. The general functionalities of Suricata and Snort [4, tab. 1].

They used three hosts (the target, the attacker, and IDS) to simulate the attacks manually without any defence techniques such as firewalls, see Figure 2 below.

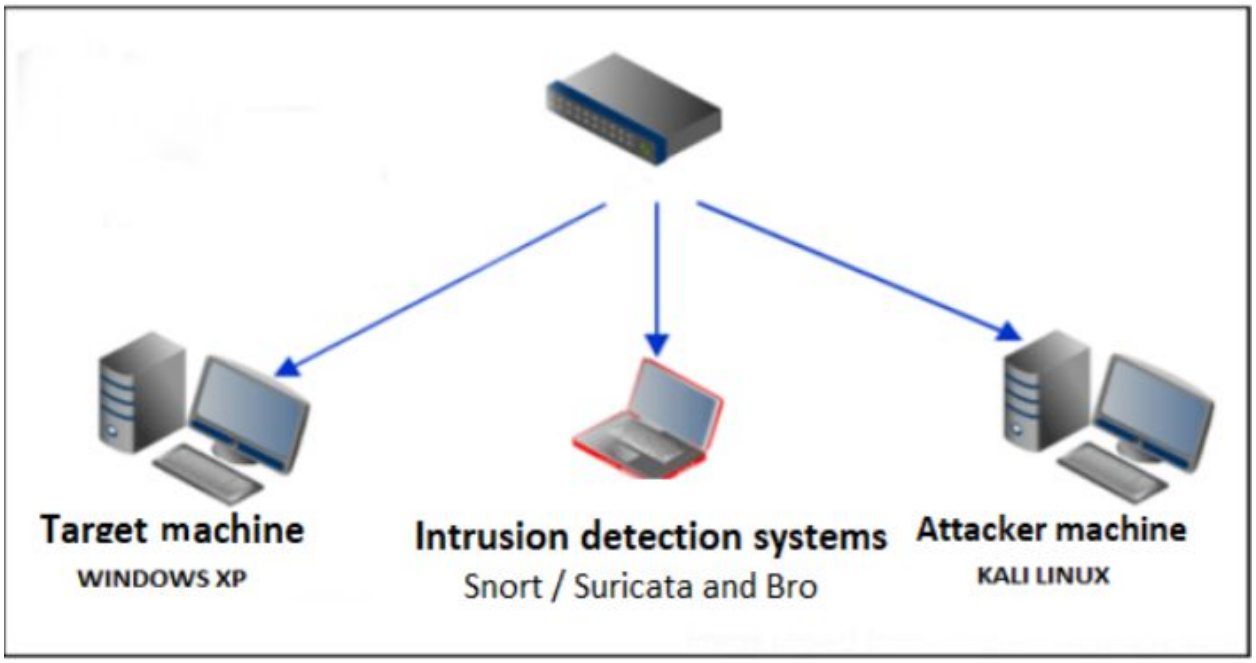


Figure 2. The implementation of the experiments [4, fig. 2].

They found that both Suricata and Snort can detect known attacks such as SQL injection and Local File Inclusion (LFI).

Besides, they tested several evasion techniques. One of them was payload mutation through Javascript obfuscation and encoding payload. Snort was able to detect Javascript obfuscation, but Suricata was not. Both were able to detect encoding payload. Also, they found that both Suricata and Snort can detect packet splitting.

Furthermore, they tested several types of DoS attacks. The first one was the ping of death which the attacker sends larger packets than allowed ones, so they can crash the target machine because the TCP/IP systems can not handle the exceeded packet size. The researchers found that both Suricata and Snort could not detect the ping of death. Another one was a syn flood attack, and they found that they can detect it and match the same signature. The third one was FTP brute force, and they found Suricata matched two signatures, but Snort matched three signatures.

In addition, they tested the detection of any malware. One of the malware was a trick bot, and the researchers found that Suricata matched eight signatures, but Snort matched just one. Another one was word docs pushing Hermes ransomware, and they found that both Suricata and Snort were able to identify this malware.

Moreover, they tested shellcode mutation and found shellcode encoded with base 64 and found that both Suricata and Snort detected it. The previous study shows that Snort and Suricata have different capabilities to detect threats.

In terms of what is similar between this peer-reviewed conference paper and our project, it has tested the ability of Suricata to detect various attacks. However, in terms of difference, it has not considered the SDN environment. We will aim to compare the results of this paper with our experiments' results to understand deeper the Suricata abilities.

4.3 A Study on SDN security enhancement using open source IDS/ IPS Suricata

This paper details preliminary research into defending against security attacks in an SDN paradigm using suricata as its primary IDS. It is a white paper presenting no results or discussion beyond the theory, however it provides useful methodologies of attack prevention that we will be able to use in our project. The architecture that this paper bases these ideas upon is shown in Figure 3 and is something that we will look to implement in our project with the appropriate (discussed in section 6) tools. The security issues that this paper theoretical explores include:

- Firewall Implementation
- Network Scan Detection
- Abnormal Traffic Detection
- Brief discussion into Attack Prevention using an OpenFlow tool [17]

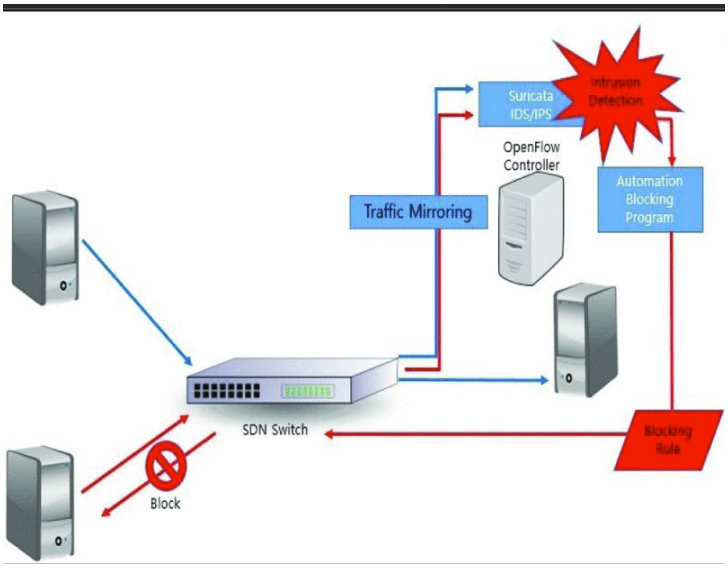


Figure 3. SDN architecture with Switch, Controller, Hosts and Suricata as IDS/ IPS [17, Fig. 3].

In terms of what is similar between this short paper and our project, it has effectively laid the theoretical groundwork for implementing advanced security detection patterns in the SDN paradigm as shown in Figure 3. However, in terms of difference as stated previously it hasn't provided any proper implementation details or experimental data. It also hasn't looked at how this system scales as hosts get larger and when there are many different sdn switches leading into the hosts of the system. We will aim to build upon this paper with our experiments and implementation.

5. Project Aim

The aim of this project is to evaluate Suricata's adequacy as an NIDS within an SDN environment. Prior research has evaluated Suricatas ability and performance outside of an SDN environment with little focus on IDS's within SDN, hence this project will look at expanding into the SDN paradigm. Furthermore, as a potential extension of this, can it be integrated with an IPS to block network attacks, as depicted in section 4.3.

In order to achieve this, we will be developing general Suricata rulesets for DOS attacks to test the effectiveness of the rulesets and Suricata as a whole. This will also require the development and simulation of such attacks, which may also facilitate external python programming. Additionally, this may also be developed in conjunction with Suricata detection if more advanced detection is required. Reference may be made to implementations outside of an SDN environment and how they compare in terms of effectiveness.

Overall the project looks at the following:

1. Simulating Different types of attacks on an SDN network:
    a. DOS Attacks
    b. Probing attacks
2. Can Suricata be implemented as an NIDS within an SDN environment?
    a. Detecting various intrusion attacks (depicted above) with varying
        i. Flooding Rate
        ii. Network Topologies
        iii. Internal vs External Attacks
3. Can an IPS be integrated into this implementation? (Potential Extension)
    a. Prevention of various intrusion attacks (Item 1) with listed variations (Item 2a)
4. Does this implementation provide real world application?
    a. Strengths and Weaknesses of Implementation
    b. How can Suricata be further developed to provide a better implementation within SDN

6. Methodology

6.1 Tools

6.1.1 Mininet

We will be using Mininet as our experimental test-bed for the experiments that we will run. Mininet is useful as it provides an instantly configurable virtual network who's API allows it to be easy to interact with and modify. Mininet also allows Suricata to be integrated within it and thus will allow us to appropriately test Suricata on different topologies and different link parameters.

6.1.2 Suricata

As explained, previously Suricata is an open-source intrusion detection system tool that much like the match-action paradigm of SDN switches, matches packets to rules and logs them in a file [3]. Another important point about using Suricata is that it cannot be directly used on top of an SDN switch. Rather a port mirroring scheme like Figure 3. must be used where Suricata is connected as another host to the switch and all the traffic coming inbound in the switch is mirrored to the Suricata as well through the network topology. Suricata rulesets are also able to be updated on the fly with the SIGHUP signal if it is required to do so from an external scripting language. One limitation in mirroring all of the traffic to Suricata is that if there are large amounts of traffic the inspection performance of suricata may go down. However, this limitation will need to be handled when we configure Suricata and when we conduct our experiments.

6.1.2 Controller

It currently needs to be evaluated whether an SDN controller will be required for our experiments or whether running commands to install flow rules through an external script is sufficient. However, if one is required in the future ONOS would be the best option as there is an easily exposed REST API for python to be able to communicate with. One potential source of limitation is ensuring that the speed of the REST API is sufficiently fast so that we can stop an attack as quickly as possible. Currently the project uses the default Mininet controller, and uses the 'os' python library to automatically install rules.

6.1.3 Scripting Language For Packet Rejection

Due to the port mirroring scheme the prevention mode of Suricata will not be useful as it will not be in the front of the network. Thus, a scripting language is required to issue commands and drop packet commands to SDN switches. Python has been chosen as it contains the most useful libraries and other packets and is a language we are all familiar with. Important libraries that will be used in our methodology include:

- urllib2 - as it is the easiest to communicate with the exposed REST API in order to query statistics of switches in the SDN network as well as ONOS controller
- json - The logged packet text is in a JSON format however it will be read in as plain text. This library greatly assists in parsing these texts and ensures constant time analysis of the data.
- os - Allows the program to open files and read text as well as crudely install rules onto the appropriate switches.
- socket - allows the team to generate a socket UNIX file which can be used by Suricata

As stated in the project aim this python tool may be required if more advanced analysis is needed beyond the simple match action paradigm especially as our simulation attacks become more sophisticated.

6.1.4 HPing3

HPing3 is a Linux based command line tool that we will use to simulate our DOS/Probing attacks. It features a wide variety of packet flooding and analysis tools that we will use to benchmark our Suricata rulesets [21]. We will use this tool to simulate different types of DOS attacks including things such as ICMP flooding, SYN flooding as well as more sophisticated methods such as IP spoofing. We will also use this tool to generate normal traffic to establish a control and to also test for false positives as regular traffic should proceed unblocked.

6.2 Methods and Approach

6.2.1 Overview

This section aims to provide the way in which we will set up our tools and link them together as well as a discussion of the process we intend to take when doing this analysis.

The approach that we will take is an iterative one that we will aim to do as many different times as possible. Due to the time constraints of the project only a few different topologies will be tested as there are different more critical variables that will need to be analysed and reported. As we iterate through this process, we will ensure that we back test and test these rulesets and DOS protection concepts in further depth.

6.2.2 Tools Integrability

As discussed in Section 4.3 Kiho et al. provides a good foundational architecture for us to implement advanced pattern recognition, however it fails to elaborate how the tools should properly link together. Thus, work has been done exploring the best way to allow for the architecture presented in Table 2 presented. After exploring the Suricata options there is an option that allows you to generate a UNIX socket for the tool to send data through, this is more efficient that model continuously analysing the file as when there is no new data, it will block rather than busy waiting. Once it ascertains whether a certain set of packets are indicative of an attack on the network it will need to robustly talk to the SDN controller to install new flows onto switches to drop packets from this IP. Using the tools described in section 6.1 this following table briefly describes how each component fits in our overall architecture.

Architecture Component	Implementation
Overall Network Virtual Environment	Provides overall test-bed for creating virtual networks
Traffic Mirroring	Using ovs-ofctl tool to delete ports and create a mirror on the appropriate interface
Suricata IDS Engine	Suricata IDS
Automated Blocking Program	Can be any scripting language which can open sockets
Blocking Rule	Currently being installed by the os.system method in python and by running the ovs-ofctl tool. In the future with ONOS if necessary

Table 2. Ways of Integrating tools for the project.

6.2.3 Method and Approach

A standard methodology for testing and documenting results should be developed to have consistency in our results. Firstly, a standard protocol for launching Suricata and setting up port mirroring has been created and distributed to our whole team. As we plan to run this setup on a virtual machine it is hoped that potential issues surrounding issues with computing and results should be minimized as the virtual machine will be allocated the same on all our machines. If there are discrepancies in our results due to the machine speed, efforts will be made to remedy these issues by running all reported experiments on one machine.

The steps the team to conduct these experiments are as follows:

1. **Develop Topologies in Mininet**

We intend to use one of 3 standard topologies namely:

- Simple - A simple 2 host and 1 switch topology that we will use to get familiar with our tools and implement the simplest attacks
- Mesh - Where most hosts are connected to each others via switches and links [19]
- Tree - Hierarchy of different switches and hosts arranged in a tree-like manner [19]

These provide a good standard baseline for us all to use and to analyse if there are any discrepancies between the difficulty of DOS analysis with Suricata in these topologies. Other topologies may also be analysed if time permits however, these will be our baseline test topologies. The size of the topology is something that we will look to expand and experiment with. On top of this extra hosts will also be added to simulate 'attackers' people that we start potential attacks from.

2. **Develop Attack Methods and Use Common Attack Methods**

Good attack simulation is a vital part of developing against DOS attacks. There are many attack types that we will look to simulate using HPing3. These common attack methods include:

1. **Syn Flooding -** In this attack the attacker sends the server a series of syn requests to use up resources on the host, this will require the host to be a TCP server in Mininet [20].
2. **ICMP Flooding -** This attack aims to overwhelm the server with ICMP or 'ping' packets as it is connectionless and can be sent from anywhere. Excessive amounts of these packets in turn overload the server and cause a denial of service where the machine is unable to function normally [20].
3. **Teardrop Attacks -** This attack sends fragment IP packets with oversized payloads which may cause a crash of the host machine's operating system [20].
4. **Distributed Attacks -** These attacks involve multiple DOS attacks, launched from many different machines which aim to overwhelm the server quicker [20].

Combined with this we also aim to emulate IP spoofing which is sending packets with fake IP addresses in conjunction with these methods in order to increase the realism of our simulated attacks [18].

3. **Develop Attack Detection/ Prevention Methods with Suricata Rulesets and Python**

In conjunction with our attack simulations, defence methods will also be implemented. This will be done using Suricata and Python and there are many ways of attack that we will look to implement. A few these include:

1. **Rate Threshold Detection -** This method of prevention is monitoring the number packets from a particular source and if this exceeds a threshold then temporarily black list this IP/MAC address from the network.
2. **Blackholing and Sinkholing -** All detected traffic that is identified as malicious gets sent to a null interface.
3. **Asymmetric Packet Handling -** Getting packets to traverse from a source to a destination in one path and take a different path when it returns to the source.
4. **Signature Based Detection -** Using a match-action scheme to search for exact or approximate packet headers.
5. **Anomaly Based -** Uses machine learning to detect deviations from normal traffic to allow for predictions when anomalous packets arise.

4. **Testing / Running Experiments and Collecting Results with Steps 3 and 4**

At this stage in our proposal we aim to the measure the following key metrics:

- **Number of False Positives Blocked**: It is imperative that our rulesets and programs do not block regular traffic from entering so it is important that we false positives. This will be measured by taking a mix of regular traffic and traffic with malicious intent and sending both to the server and measuring if any regular traffic gets blocked.
- **Time of Response of DOS Attack Detection**: We aim to examine how quickly our defensive tools work on our simulations especially as they get more sophisticated. This will be done by printing out the timestamp when the attack launches and print out the timestamp at the first sign of detection and taking the difference.
- **Delay between detection and prevention**: Attack prevention is the last factor and it is important to measure the discrepancy between when we detect a threat and when the first rule to drop packets gets installed.

Lastly, this will be an iterative process and thus we aim to present in our final report the many iterations of the Suricata rulesets that we have developed. This will be coupled with more advanced detection from Python as well as discussing how our attacks got more sophisticated to continuously improve our defence systems.

7. Plan

See the Table 3 below for a comprehensive outline of our team's project plan split into a week by week timeline with its major tasks and milestones.

Timeline	**Tasks and Milestones**
Week 1 (3/08 - 9/08)	Introduction to advanced Computer Networking topics (SDN, Queueing Theory, P2P Networks, Wireless Networks, Quality of Service, Network Management).
Week 2 (10/08 - 16/08)	Team formation and ice breakers; Set up team collaboration tools (Messenger for text communication, Zoom for virtual meetings, Google Drive for file sharing).
Week 3 (17/08 - 23/08)	Decide on the project topic (Exploring Suricata); Install relevant software tools, e.g. Suricata, Python, VM, etc.
Week 4 (24/08 - 30/08)	Work allocation for Progress Report (documented in a Google Sheet, https://docs.google.com/spreadsheets/d/1UeRkFGDsUxPWgIT_ZKLFT0Xs6X3xaHMEsukGT9WegWM/edit?usp=sharing).
Week 5 (31/08 - 06/09)	Background research; Progress report writing; Running preliminary DOS experiments with port mirroring.
Week 6 (07/09 - 13/09)	Discussions on finalising progress report; Generate Standard Topologies; Simulation of SYN Flood; Project Proposal (11/09).
Week 7 (14/09 - 20/09)	Simulate DOS attacks in Mininet with Suricata 'listening in'; Configure Suricata with rules to detect and prevent attacks; Gather results from experiments.
Week 8 (21/09 - 27/09)	Interpret experimental data and transfer findings into the final report.
Week 9 (05/10 - 11/10)	Final report writing and prepare for project presentation.
Week 10 (12/10 - 18/10)	Practise presenting and finish off the report; Project Presentation (16/10).
Week 11 (19/10 - 25/10)	Project Reflection.
Week 12 (26/10 - 01/11)	Final Exam preparation.

Table 3. Project work plan.

8. Progress so far

8.1 Overview

Throughout the first 5 weeks of this project, our team has been working on a number of things to set ourselves up for our project proposal and beyond. We have completed extensive background research into relevant networking topics such as: network security practices, SDN and its associated concepts, and sample related work for Suricata. Having completed this research, our team has gained a much better understanding of each individual concept on a high level and how they can all fit together to achieve the goals of our project proposal.

One initial problem we had was properly configuring Suricata on our Linux virtual machines. Fortunately, one of our team members was able to correctly configure Suricata and run basic experiments. After doing this, he was able to export the current state of his VM and share this with the rest of the team via upload to Google Drive. This was an excellent solution, as everyone was easily able to be caught up to the same page and participate in exploring Suricata for themselves and contributing experimental results to the overall project.

8.2 Experimental Progress

As part of our preliminary experiments, we could observe that the Suricata IDS component logs based on a certain set of rules to a file called eve.json and then the log can be analysed to inform the controller to add flows, i.e. drop packets from certain IPs. This process was done with a python script which analyses the log continuously and then rules are added to the switches to block packets for certain amounts of time whilst keeping a backlog of blocked threats.

We were also able to perform a Basic DOS Protection Experiment as part of our preliminary testing. This experiment involved an SDN switch receiving an X number of packets from a source then dropping packets after that. As can be seen in Figure 4 below, if the difference between two packets is less than a second, a drop rule is installed on the switch. Hence, the ping receives 6 packets and loses 40%; the rules get installed on the 6th packets. The first two packets haven’t been logged.

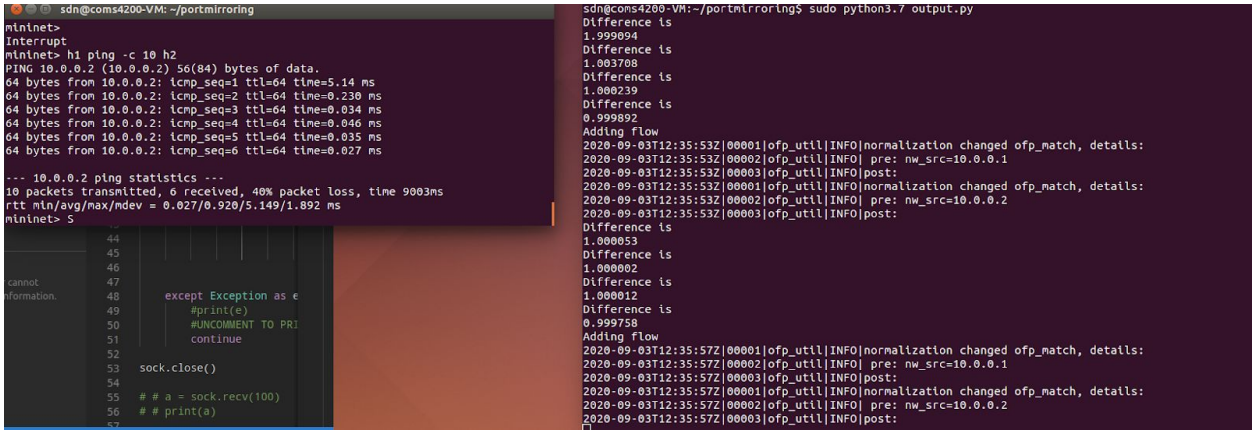


Figure 4. Screenshot of preliminary Suricata experiment.

From all the preliminary experimenting, we realised continuously analysing logs was time and resource intensive. Instead, you can have the controller connect to a socket and send data through that, rather than having to write to a text file and read it over again. This can be done using a python script and editing the rules in suricata.yaml.

For the final report, we hope to go from using simple topologies to applying the methods in our preliminary experiments to more advanced topologies such as tree topologies with our own custom analysis along with the packet logging of suricata. We aim to also implement barriers against as many common intrusion attacks and to implement under this SDN and Suricata paradigm. We plan on trying to get our virtual network working with the ONOS controller and get the controller to output logs in the json format for easier analysis. Our findings will definitely be new and interesting to see how IDS/ IPS such as Suricata can be configured and applied to University and enterprise networks.

8.3 Project Plan Tracking

As seen in Section 7, our team has created a detailed work plan on the key timeline and milestones for this project. We have been following that quite well and are overall tracking well towards our project goals, having completed all key tasks up until the current point in time. Due to the unprecedented times we live in, it is difficult for us to meet and discuss project related topics in person. However, we have all accepted our circumstances and adapted to the situation. We have weekly meetings on Zoom to discuss our progress and any daily issues we've been having we would discuss over text in Messenger. Overall, having completed our research on Suricata, and run some basic experiments, we have got a clear idea of how we want to tackle and finish off this project. As stated before, we are on track with our proposed work plan. We will continue to work hard with strong initiative and self-reliance towards our end goal.

9. References

[1] "Use offense to inform defense. Find flaws before the bad guys do.", Cyber-defense.sans.org. [Online]. Available: https://cyber-defense.sans.org/resources/papers/gsec/host-vs-network-based-intrusion-detection-systems-102574

[2] J. White, T. Fitzsimmons and J. Matthews, "Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata", in The International Society for Optical Engineering, 2013.

[3] "Suricata", Suricata. [Online]. Available: https://suricata-ids.org

[4] O. Bouziani, H. Benaboud, A. S. Chamkar, and S. Lazaar, "A Comparative study of Open Source IDSs according to their Ability to Detect Attacks," in Proceedings of the 2nd International Conference on Networking, Information Systems & Security, 2019, pp. 1-5.

[5] "8.3M records of Freepik and Flaticon users stolen in SQL injection attack - SiliconANGLE", SiliconANGLE, 2020. [Online]. Available: https://siliconangle.com/2020/08/24/8-3m-records-freepik-flaticon-users-stolen-sql-injection-attack

[6] B. Hellard, "UK college hit with DoS attack on results day", Itpro.co.uk, 2020. [Online]. Available: https://www.itpro.co.uk/security/distributed-denial-of-service-ddos/356849/uk-college-hit-with-dos-attack-on-gcse-results

[7] "Intro to Information Security | Udacity Free Courses", Udacity.com, 2020. [Online]. Available: https://www.udacity.com/course/intro-to-information-security--ud459

[8] O. N. Foundation, "Software-defined networking: The new norm for networks," ONF White Paper, vol. 2, no. 2-6, p. 11, 2012.

[9] R. Maaloul, R. Taktak, L. Chaari, and B. Cousin, "Energy-aware routing in carrier-grade ethernet using sdn approach," IEEE Transactions on Green Communications and Networking, vol. 2, no. 3, pp. 844-858, 2018.

[10] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008.

[11] M. Team, "Mininet Overview - Mininet", Mininet.org, 2020. [Online]. Available: http://mininet.org/overview

[12] M. Pihelgas, "A COMPARATIVE ANALYSIS OF OPEN SOURCE INTRUSION DETECTION SYSTEMS", Master's Thesis, TALLINN UNIVERSITY OF TECHNOLOGY, 2012.

[13] R. Fekolkin, "Intrusion Detection and Prevention Systems: Overview of Snort and Suricata", Internet Security, A7011N Luleå University of Technology, 2015.

[14] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "Denial-of-service attack-detection techniques," IEEE Internet computing, vol. 10, no. 1, pp. 82-89, 2006.

[15] V. Nagadevara, "Evaluation of Intrusion Detection Systems under Denial of Service Attack in virtual Environment," Faculty of Computing, Blekinge Institute of Technology, Karlskrona Sweden, 2017.

[16] C. GARCIA, "Suricata IDS: an overview of threading capabilities", AT&T CYBERSECURITY, 2019. [Online]. Available: https://cybersecurity.att.com/blogs/security-essentials/suricata-ids-threading-capabilities-overview

[17] K. Nam and K. Kim, "A study on sdn security enhancement using open source ids/ips suricata," in 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018: IEEE, pp. 1124-1126.

[18] H. Dalziel, "5 Major Types of DOS Attack | Hacking Tools & Growth Marketing Tools", Hacking Tools & Growth Marketing Tools, 2020. [Online]. Available: https://www.concise-courses.com/5-major-types-of-dos-attack

[19] 1. (CheckList) and F. License!, "5 Major Types of DOS Attack | Hacking Tools & Growth Marketing Tools", Hacking Tools & Growth Marketing Tools, 2020. [Online]. Available: https://www.concise-courses.com/5-major-types-of-dos-attack/. [Accessed: 10- Sep- 2020].

[20] "Types of Network Topology in Computer Networks | Studytonight", Studytonight.com, 2020. [Online]. Available: https://www.studytonight.com/computer-networks/network-topology-types. [Accessed: 10- Sep- 2020].

[21] "hping3", Tools.kali.org, 2020. [Online]. Available: https://tools.kali.org/information-gathering/hping3. [Accessed: 10- Sep- 2020].

[22] Csie.nqu.edu.tw, 2020. [Online]. Available: http://csie.nqu.edu.tw/smallko/sdn/ids_ips_suricata.htm. [Accessed: 06- Sep- 2020].

[23] "How to Configure & Use Suricata for Threat Detection", Infosec Resources, 2020. [Online]. Available: https://resources.infosecinstitute.com/configure-use-suricata-threat-detection/#gref. [Accessed: 06- Sep- 2020].

[24] "Lessons for the Enterprise from Running Suricata IDS at Home - VDA Labs", VDA Labs, 2020. [Online]. Available: https://vdalabs.com/2018/06/18/lessons-for-the-enterprise-from-running-suricata-at-home/. [Accessed: 06- Sep- 2020].

[25] "Suricata – Vuurmuur Firewall", Vuurmuur.org, 2020. [Online]. Available: https://www.vuurmuur.org/trac/wiki/Suricata. [Accessed: 06- Sep- 2020].