ISSN No:-2456-2165

Edge Based Network Attack Detection Using Pulsar

Sheetal Dash

ISSN No:-2456-2165

ABSTRACT

The edge-based network attacks are increasing largely in size, scale and frequency with the booming internet. The Distributed Denial of Service (DDoS) attacks is one of the most diffused types of attacks in the cyberworld which is a great concern for all organizations today. There were 5.2 billion Google searches in the year 2017 alone. Research shows that there cannot be a better example to show how prevalent internet use is nowadays. What started off as a point to point conversation in 1970s as a mode of communication has expanded to millions and millions of devices communicating with each other via some or the other form of the web. And therefore, this research focuses on the strategic approach that has been taken to defend from the growing cyber threat.

In this dissertation, the focus is on defending against these edge-based network attacks and collaborating with other neighboring networks in order to communicate with them to transfer information about potentially malicious hosts. This research has focused on exploring ways of integrating the Software Defined Networking with the pub-sub messaging system in order to show a collaborative approach of defense to these attacks. The attacks from unknown multiple sources have also been analyzed in order to cope with them through this robust solution that has been proposed and implemented in this dissertation.

ISSN No:-2456-2165

ACKNOWLEDGEMENT

I would take this opportunity to extend a heartfelt thanks to my supervisor, Dr. Sandra-Scott Hayward for providing an opportunity to be a part of this research and guiding throughout the time over completion of this work. Her experience and expertise have been very helpful in supervising my work. She has constantly been a helping hand at all times to my work at Centre for Secure Information Technologies (CSIT), Queen's University for this research which provides the basis of this article. I am very thankful for the valuable time she has put in to assist me in this research.

I would also like to thank my family for their constant support during my academic career and helping provide me with the opportunity to study at Queen's University. Last but not the least, I would like to thank God for supporting me while I pursued my higher studies and for always being there for me at all times.

Table of Contents

List of Figures

List of Tables

ISSN No:-2456-2165

CHAPTER 1
INTRODUCTION

Emerging technology trends, such as the concept of the Internet-of-Things (IoT) or the advent of the Industrial Control Systems (ICSs) with the internet, raises questions about the challenges that need to be tracked from a security perspective. The author's preliminary work starts off by analysing the existing architectures and later goes on to the novel topics of security and privacy.

With the boom in the market for IoT-based devices, (which is evident from the surveys that have been performed), it is imperative to expect that the vulnerability to cyber-attacks has also increased manifold.

Nowadays, Intrusion Detection Systems (IDSs) play a key role in any systems preparedness to defend in case of an attack. They have been studied in depth in the past and used widely. However, the known drawback to traditional IDSs is its non-scalability to large networks or highly distributed denial of service attacks. Collaborative IDSs (CIDS) [2] have come into the picture in an attempt to cover these shortcomings. The monitoring components in these systems collect and exchange data in a collaborative manner. Depending on such kind of architecture, the proposed solution of the work in this thesis has been focused upon.

Goal

The main objective of this research is to detect a DDoS attack in its early stages itself and notify the hosts nearby to block the malicious traffic from the attacker. The term 'early' depends on the network on its own, the tolerance of the device and the properties of the network traffic. Having said that, the sooner the detection and mitigation happens the better it is. Waiting longer could risk the device getting inundated with packets from a malicious source. In addition, the Internet Protocol (IP) addresses after getting detected would be blocked and the same would be notified to the neighboring networks. The detection and mitigation of the attack collaboratively without minimizing the persistent performance are one of the objectives of this work.

The focus of this dissertation is to present a detailed insight into collaborative implementation techniques of detecting the edge-based network attack, prevent them, followed by their execution and implementation in a running network. In addition, the detected attack must be communicated to other neighboring networks and notify them. The author shall even carry out a detailed analysis and research in order to work out a detailed mechanism of the same.

Research Objectives

The novel character of the Collaborative Intrusion Detection System (CIDS) area results in many research questions and limitations that will be in depth discussed in this dissertation. Briefly, this field unites research from many different areas of networking as explained in the section of related work in this dissertation.

This thesis is attempting to answer the following research challenges for Collaborative IDSs: -

- Detect an attack before the target host or the controller is inundated with a huge number of malicious packets.
- Propose a strategic mechanism that performs the sharing of information with minimal communication overhead.
- Effectively exchanging data to other networks, in the context of a collaborative attack detection framework.
- Impact of such a framework – performance, testing (positive and negative scenarios).

First, an efficient, accurate and yet speedy attack detection method needs to be formulated. Furthermore, handling the situation of other neighboring networks being affected in an organization, that is realistic enough to be utilized for the evaluation of an organization's security perspective, will be a significant value add to the existing research.

Another main aim of this research focuses on detecting the attack as soon as possible, preferably at the start of its launch. This will allow the controller to bar the attacker from accessing the network before the attacker could render the target unreachable.

In order to achieve this objective, a high-speed and highly effective detection method is needed that works with the Software Defined Networking (SDN) architecture. The framework must be lightweight, else it shall hamper the performance. Also, the framework must effectively notify adjacent networks about the attack and prevent them from being a victim.

Contribution

The main aim and objective of this field of research are to detect and mitigate DDoS attacks in a collaborative manner. However, the degree and the rate of the detection varies from one approach to another depending on the algorithm and environment used. After analyzing a lot of papers, it is found that not all researchers have found a profound solution to it. Rather, there are a few areas which need to be focused upon to deal with such kind of an attack. The following can be a few that can be contributed to this field of research:

- Detect attacks from known as well as unknown sources.
- Blocking malicious traffic and letting genuine traffic pass through as opposed to periodically dropping all packets forwarded to the victim when a threat is detected.
- Share information about the attacker to other networks and eliminate the risk before it corrupts the whole organization's infrastructure.
- Calculate the accuracy and sensitivity of the attack very minutely to verify the contribution.

The aim of the research is to build a solution in this framework considering all the above points to assist in the detection and mitigation process. In addition, this approach can be compared with few of the existing models and can be identified as to how this framework would be better for the organization.

Structure of the Thesis

The first few sections of this work presented is the initial step of the overall architecture, that is to effectively detect DDoS traffic at all the nodes near the edges and block them. This thesis includes the following chapters and is organized as follows:

- Chapter 1 reviews the background information of the DDoS attacks and other related approaches used in this field of research as well as in the design and implementation of the framework.
- Chapter 2 gives a basic idea of the works that have been researched in this field to deal with such attacks from the very early stage. To effectively fight against these attacks, it is very necessary to know the cause of the attack or the root from which it starts. For example, if the attack can be identified at the edge of the network and can be distinguished as a malicious flow of traffic, then this malicious traffic can be filtered for defending the attack.
- Chapter 3 discusses in details the aims and objectives of the work with the open challenges that are analyzed as a part of this work. In addition, it even provides the justifications of the choice of choosing the technologies and the tools used.
- Chapter 4 provides the design model of the framework that has been discussed in order to detect, mitigate and communicate with the other neighboring networks to protect them even from the DDoS attacks.
- Chapter 5 describes in detail the environment that has been set up in order to implement this framework

including the analysis of the traffic and implementing ways to mitigate them.

- Chapter 6 describes and shows the collaborative approach by using the pub-sub messaging system. It evaluates the approach by checking its performance, accuracy, sensitivity and even compares it with other existing approaches justifying the reasons for it being better.
- Chapter 7 discusses further scopes of this framework as the future part of this research.
- Chapter 8 summarizes this thesis including all the important points in this research.

ISSN No:-2456-2165

CHAPTER 2
REVIEW OF RELATED WORK

Introduction

This chapter of reviewing the related work aims to introduce the reader to the various topics that shall be discussed further down in the rest of the thesis. The first section discusses SDN Architecture which is being a central part of this dissertation, including some initial classifications and definitions related to them.

The second part of this section deals with certain collaborative intrusion detection frameworks which is an additional framework that can appreciably improve the existing mechanisms to detect attacks.

Lastly, this chapter deals with the topic of evaluating the Intrusion Detection System (IDSs) with importance on the corresponding datasets. At the same time, the importance of the tools and mechanisms for creating such datasets will also be evaluated. As it shall soon become clear in the course of this thesis, the latter topic of the collaborative evaluation mechanisms stays as a big challenge in this research area.

Mechanisms which uses Devices against the Victim

DDoS attacks have become a weapon of choice for hackers, intruders and cyber terrorists [5]. These attacks can swiftly weaken a victim, causing huge financial and reputational loss. A lot amount of work and research is being carried out in this field and a lot of techniques have been identified in order to deal with them. This section provides background information on the DDoS attacks, its detection mechanisms and a few related works on how they have been implemented.

DDoS attacks are made possible by the Internet. The term distributed as the name suggests describes that such kind of a distributed detection framework detects the DDoS flooding attacks by monitoring the flow of packets in the network, of the abrupt traffic changes at various distributed network points.

The forwarding elements within the Internet forward packet towards their destination with a very little or no consideration towards the behavior of the sender to the receiver. Hence, Mirkovic and Reiher[4] presented taxonomy on the forms of DDoS attacks, with an observation that the Internet was not designed to defend traffic. The simplest being a flood attack is characterized by a victim being sent tremendous volumes of traffic to consume resources. There were two other types, one of it requiring the attacking party to send malformed packets to machines that cause applications to freeze and the other requiring the attacking party to gain privileged access to devices within the victim's network. Consequently, these devices are then used against the victim until the target resources become unavailable. In addition, they have also identified various approaches to detect such kind of attacks which are pattern-based, third-party detection and anomaly detection.

Pattern Detection used signatures of known attacks to identify DDoS attacks as traffic passes through a network. These signatures include a combination of IP addresses and port numbers, which are used in Static Classification (an example of such kind of a detection mechanism). Snort [20] is an example of the intrusion detection system which is based on signatures to identify malicious traffic in the network. However, these kinds of techniques are not found to be flexible as they must be updated regularly to identify new attacks.
Third-Party Detection basically relies on external networks to detect and inform them about the occurrence of an attack. The limitations of such kind of a detection approach are that the detection signal must travel to the victim network. But, it would be difficult or may not be possible for the signal to make it to the victim network. Although an approach of having a separate network for signalling would be helpful, it may be of a higher cost to the victim.
Anomaly Detection model creates a normal traffic behavior and then compares it against the flows in the

network as traffic passes through. These detections systems are also useful in other domains to identify faulty equipment in critical systems. It is important to distinguish anomaly detection from other similar processes handling outliers in data, which are noise removal and novelty detection. Noise removal is the process of removing unwanted data points before the analysis of data. On the other hand, novelty detection helps in identifying previously unobserved behavior which is then included in the model as the behavior to be a "normal" one. However, they tend to misclassify traffic which is not malicious, called false positives. This can be very inconvenient for the legitimate users of a resource as they may be flagged and treated as an attacking party. Hence, these systems can be helpful to identify cases that fall outside the bounds of normal behavior.

SDN Mechanisms

The mechanism explained above indicates that there are a lot of various approaches identified by researchers to identify DDoS attacks collaboratively. This section covers some of the core topics, related researches, and background within the domain of SDN. This would be helpful and would be used throughout the thesis.

SDN/OpenFlow has been integrated with classification techniques for classifying network in various scenarios. The use-case for detecting DDoS attacks collaboratively in this thesis is important, to show that this kind of a model can be used in integration with a pub-sub messaging system.

Classification mechanisms traditionally operate on the controller. Recent researches have seen this task to be moved closer to the switches in the data plane. Lin *et al.* had come up with a proposal that the processing overheads imposed by the detection algorithms and the classifiers on a controller can be removed by transferring the function to some other device in a reliable manner [9]. However, this kind of transfer was exemplified to be scalable as well as a necessary one.

The SDN based platforms for classifying malicious and legitimate traffic are typically evaluated in virtualized environments. There is room for review and challenge that such a mechanism does not accurately show how a system would work in a real-time environment despite being the convenience of the environment. Hence, this has been tackled by the proposed solution that has been explained further in this research, which even utilizes the maximal amount of traffic to maintain a good performance level. This piece of research even utilizes SDN integrated with Pulsar in order to distinguish the malicious and legitimate traffic at the host level itself. The SDN platform detecting and blocking the malicious traffic at the host level demonstrates resiliency to the attacks right from the start and prevent overwhelming of the traffic at the end. Consequently, it would also communicate to the other neighboring networks about the existence of the attack from the particular host and even notifying to block them from the network.

OpenFlow is an implementation of the Software Defined Networking (SDN) Architecture which will be used further in this thesis. It is characterized by the separation of the control and data planes.

The logically centralized control-plane offered by SDN has been considered a very convenient approach for detecting the DDoS attacks which have been further explained in this work. The data plane is responsible for deciding how traffic is forwarded through a network and is typically realized by a logically centralized controller. The OpenFlow manages network traffic through the definition of flow table entries which are stored in a switch's flow table. These entries in the flow table depict how packets with matching characteristics are handled by the switch [36].

In OpenFlow, the switches within the network are connected to each other, receiving instructions on how to forward packets through the entries in the flow table. As a result, they can provide the controller with the information as necessary. The higher level of such kind of a model discusses that the business applications

communicate with the services in the network and these network services then enforce the desired behaviors on the infrastructure.

There have been research efforts around implementing the flow-based security for IoT Devices using a Software Defined Networking (SDN) Gateway. Austin [2] et al. proposed the use of this gateway as a distributed means of monitoring the traffic in the network which is originating from and directed to IoT devices. This has been devised in order to provide flexible and secure integration seeing the exponential growth expected in the number of IoT based devices. The SDN Gateway proposed has the ability to both detect abnormal behaviour and perform a programmed response, i.e. either blocking or forwarding. This is a new kind of adaptive flow-based security mechanism which performs real-time analysis of network traffic from IoT devices to detect anomalous behaviour. Hence, there is room for further research in order to deal with such challenges.

Centralized Detection Mechanisms

Further, with the advancement of the technology came the distributed framework for the detection of the DDoS attacks. Snapp et al. [22] proposed one of the earliest centralized CIDSs known as the Distributed Intrusion Detection System (DIDS). The author, in this work, detects the malicious traffic over the network that is being monitored and creates a comprehensive result of its security state. This kind of a framework unites distributed monitoring with a centralized data analysis and contains the elements like DIDS director, host and network monitors. The DIDS director represents to be the central analysis unit. The detection techniques used in this model would not be able to compete against a sophisticated attack. Therefore, this model's accuracy is deemed poor. In addition, there is a lack of self-configuration mechanisms in such a model. The communication and computation overhead in this approach increases with an increasing size of the monitored network, hence they are not scalable. This is one of the open challenges that need to be tackled in the case of a distributed framework.

Further, cooperative intrusion detection framework (CRIM) was introduced in [23]. This is a centralized cooperative framework which connects to isolated IDSs and acquires its data. Based on that data the framework raises alerts. After that then an alert clustering function generates clusters of alerts based on a relation of similarity [24].

Cross-Layer Approach

In this kind of a methodology, the items at various layers communicate and transmit information with each other. These approaches work on the SDN platform by integrating with it and providing an easy access to the network status data. Wang et al. [25] have proposed a cross-layer mechanism to configure the underlying network at runtime depending on the dynamics of the big data application. The author here uses high configurability of the SDN switches and even the scheduling approaches of the Hadoop job in order to accommodate the dynamic configuration of the network in a hybrid network with Ethernet and optical switches. In this paper, the results of the analysis were the improved performance of the application and network utilization with a low overhead of its configuration.

Traceability

Few researchers have thought of tracing the origin of the DDoS attacks first and then mitigating them. Savage, Wetherall, Karlin, and Anderson [26], have introduced a trace-back mechanism using the probabilistic packet marking algorithms to reduce the strength of DDoS attacks. This is a mechanism introduced which helps the victim to recognize the path of the attack without any technical involvement from the Internet Service Providers (ISPs). When there is a lot of traffic or attacks instigated from various distributed locations, the packets travel from one network to another through the intermediate routers and are checked against the access tables which are to be forwarded to allocated networks without packet

modification. However, the researchers suggest header modifications on each IP header with specific flags at each intermediate router before the packets reach their next hop. When a DDoS attack is initiated, the victim's network is populated with a high volume of packets resulting in slow network traffic and poor response. Consequently, at this point, the system calculates the packets that are received by the operating system's Kernel and reconstructs the path of the attack using marked values assigned by each router. Thereby, informing the closest routers to limit their traffic rate to minimise the strength of the attack.

Yaar, Perrig, and Song [27] have proposed a marking mechanism called Stack Path Identification (StackPi). In this mechanism, the authors concentrated on detecting spoofed IP addresses and tracing-back the source of the attack. This approach includes two marking methods, Stack-based and Write-ahead marking, which helps in substantially increasing the performance according to them. It also states that such kind of a methodology almost completely eliminates the effect of legacy routers. In addition to this work, Beak, Lee and Kim have developed another kind of marking technique using Link-ID to construct the path of the attacking packets. It has been claimed in this approach that the Link-ID information between the Border Gateway Protocol (BGP) routers provides more accurate results than marking IP headers with randomised values. Link-ID is the information path between the BGP routers in any Autonomous Systems (AS) and BGP router provides Link-ID information on the flooding packets. As soon as the victim receives the flooding packets, it constructs the packets and then calculates the path of the attack using the Link-ID values.

Law, Lui, and Yau, [28] complemented and enhanced the probabilistic marking method by more accurately identifying the locations of the DDoS attackers. This is performed to allow the victim to infer the local traffic of all the routers that the DDoS attack passes through. The rate of the traffic can be identified through a framework where the rate can be determined in unit time. In other words, it can be described through an example as such that, if a victim discovers its sides to be under DDoS attack, it requests all its known routers to mark the incoming packets with any given probability.

P. Based on the marked packets the victim receives, a graph can be represented showing the origin of the attacks. This reduces the traffic based on the router with the highest number of marked packets.

Entropy Based Mechanisms

Further, Yu, Zhou, Doss, and Jia [29] used the method of entropy variations methodologies to identify the launch of the attacks. When there are no attacks within the network, the routers record the entropy variations of the local flow of traffic. In this approach, flow and entropy variations have been used in order to detect the attacks and once it has been detected, a pushback process has been used which identifies the location of the zombies. Based on this entropy variation, each router has accumulated enough information that identifies the upstream routers and then submits to the immediate router in order to identify the flow of traffic. This flow of traffic is based on local entropy variations that routers monitor. The intermediate respectively routers forward requests to further upstream intermediate routers to find the source of the flow.

Furthermore, Su, Wu, Hsu, and Kuo [31], on the basis of the performance of the network introduced a trace-back and mitigation method. The author claims that this is based on the monitoring packet loss and even the rate at which packets arrive. This kind of a methodology does not analyze the traffic post-mortem rather it uses online analysis. The packet filtering mechanism is proposed to reduce the DDoS attacks and Intrusion Detection Systems (IDS) are also used to support all edge routers. These routers, in turn, support Simple Network Management Protocol (SNMP) [32]. Such kind of a framework on the edge of the routers conducts two phases, one being the DDoS attack trace-back and the other being the attack mitigation. In order to discover the flow of the traffic, the system uses estimated attack entry to monitor the loss of packets and the arrival rate of packets.

Other approaches that have been initiated by the researchers are to intelligently make decisions to trace-back the cause of the attack. Intelligent Decision Prototype (IDP) is a supervised machine learning approach which is introduced in [33]. This kind of methodology is divided into two phases- one phase being called the pre-marked decision where packets are subjected to DDoS attack attribute analysis. If packets are genuine then they are forwarded to the next closest router and if not they are marked as not genuine. The second phase of this approach is to identify the path of the attack.

Statistical Analysis

Network analysis methods, such as Entropy and Chi-Square techniques which are quite common mechanisms, have been proposed in [34] to detect a change in the network traffic. Packet headers are represented by Entropy as independent ones which have a unique probability of occurrence. In this kind of an approach, there is a pattern evolving for every type of packet header and high standard deviation from the average limits would raise alerts about spotted anomalies. This method has its potential use in SDN which is explored further in later chapters as its quite helpful for the solution that is proposed in this thesis.

Further, machine learning and cognitive detection approaches have also been proposed for the detection and defence against intrusion. In this kind of an approach, based on the events in the network the machine is trained to continuously update a certain filtering criterion rather than setting fixed criteria on filters. One of the examples of such kind of a framework is neural networks [35] which contains many nodes working parallelly for processing the data. When these are trained and a large amount of information is provided to them, the knowledge of these aggregated nodes develops a particular pattern for the processing of similar information. The knowledge gathered over a period of time become the base of the decision-making process for the neural networks.

Cloud Mechanisms

OpenFlow has been a common field of research in the cloud networks and data centers, that are currently on the rise. There is research around resource control, improving the scalability of the network using OpenFlow and network virtualization in OpenFlow [37], the advantages of which will be explored further in this thesis. The cloud networks often deal with distributed hosts all around the world or from various locations. Quite a lot of controllers shall be needed to deal with the scalability issue we shall face if the SDN architecture is applied to cloud networks. This is explored further in order to handle this situation.

In this paper [24], a cooperative design was proposed in order to analyze alerts and generate more alerts. The main functions, one being the alert base management, alert clustering and the other being the alert merging has been suggested. In this approach, the alert manager stores alerts in a relational database in order to analyze and compare the alerts more. Further, the alert clustering analyses the alerts and even performs cluster generation for similar alerts. And lastly, the alert merger basically allows refining the clusters in order to bring an accurate alert. This approach even shows correlation by using explicit correlation when the security administrator is able to find some connection between events that it knows and even though the implicit one where the data analysis brings out some kind of mappings between events.

Further, in order to deal with such kind of attacks the new architecture of SDN proved to be one of the solutions for the cloud networks. Shin et al. [38] used OpenFlow which monitors the network traffic. The main objective here is to modify the packet flow by using the OpenFlow protocol. The motive behind doing that was to direct the packet flow in the route where the Network Intrusion Detection System (NIDS) is installed. In this research, SDN architecture shall be explored which does not need the relocation and reinstallation of the NIDS devices like the former approach needed. OpenFlow used here helps in processing the packets and adding the flow rules to the hosts and switches to travel through this path that is being monitored. This is quite useful which has been further used in this work. In order to find the shortest and

secure path for the packets, algorithms have been proposed in this paper and each of them is classified by the controller based on the time it takes to find the same. In this analysis, it has two modules, one being the SDN controller which calculates the shortest path to a monitored channel in the network and the other being the routing rule generator which identifies the topology of the network, thereby collecting the status and statistics of the network to pass it to the controller. In this case, the NOX controller is used which serves as the network control platform.

Furthermore, Xing et al. [39] proposed adding OpenFlow to SNORT [21] which is a tool for the detection of attacks in the cloud, which even reconfigures the network when there is an attack spotted by the SNORT. The approaches explained, show the capability that SDN has to be an intrusion detection and mitigation framework in a cloud network.

With the attack coming from numerous vectors from all directions, a single isolated instance of monitoring to detect any intrusion would most certainly be ineffective. To overcome this, an approach was taken, referred to as the Collaborative Intrusion Detection System (CIDS). In a CIDS, there are multiple monitors capturing data from multiple sensors and communicating between themselves in some way to raise alerts about identified botnets and blocked devices/IPs which forms to be an effective method in detecting and preventing such attacks.

Netbiotic is one of the distributed CIDS based on the JXTA peer to peer [40] framework. The focus of this kind of a framework is just not on detecting specific attacks but it even creates a faster network of interested peers which help in the exchange of the alert information. Therefore, their main goal lies in protecting the participating peers, such as by detecting rapidly propagating malware. Vlachos et al. have proposed the Netbiotic peer which works as a monitoring and analysis unit, which even hosts a notifier and a handler component. In this kind of a mechanism, each peer is accountable for identifying whether a virus is propagating through the network apprehensively along with automatically dispatching of warnings and information to other peers and even taking specific preventive measures for protecting their host by automated tightening of their security controls during this infestation.

The remaining chapters demonstrate the open challenges which are being identified to act upon, how SDN and Network Function Virtualization (NFV) along with few other tools were chosen to be evaluated on a physical network testbed and the tasks to be performed as a part of this research. Further, it even provides a broader insight on how the SDN approach has been integrated with the pub-sub messaging system, which collaborates with other neighboring networks to publish the attacker and presents the final conclusions with the future work.

Summary

To summarise the centralised framework of CIDSs has the ability to provide the maximum accuracy but they are not scalable. Therefore, such kind of an architecture can be only used for protecting small infrastructure networks. In order to deal with large organizational networks, there have to be more scalable solutions implemented. The decentralization of CIDSs seems convenient but are unsafe from attacks and even have the level of accuracy as very low than the centralized systems.

In addition, performance seems to be a challenge in these kinds of infrastructures as it depends highly on the used building blocks for interconnection and collaboration.

To date, researchers found the occurrence of the DDoS attacks and its mitigation at the switch level in the SDN architecture. Therefore, especially in this area, certain challenges need to be addressed by detecting and mitigating the attacks at the host level itself.

ISSN No:-2456-2165

OpenFlow Networks mechanisms were also discussed. The occurrence of DDoS at a very high level of frequency may create an open challenge for SDN and it can be focused on just the controller getting overwhelmed, which in turn can bring down the operating system. This states that OpenFlow has all the components that are deemed necessary for a successful detection. Therefore, the following chapters in the thesis focus on the utilization of the components of OpenFlow for the early detection and mitigation of DDoS attacks within an SDN environment. This thesis even uses these elements and measures the rate of detection in order to address a few challenges in this area of cybersecurity.

ISSN No:-2456-2165

CHAPTER 3
AIMS AND OBJECTIVES

From the above chapter, the author summarises a few models that have been introduced by researchers and their main objective behind it was to reduce the strength of the DDoS attacks. Researchers have focused on traceability while others have focused on detection and mitigation of attacks using various machine learning approaches, algorithm or statistical to increase the detection rate admitting the fact that it is almost of a small amount in all cases. The author has explained all the approaches to have an idea about the state of the art in this particular field and then used this to compare the results of the approach based on its accuracy, scalability, and performance, in order to highlight the contribution towards the work. In this chapter, there is an explanation and overview of the open challenges that are relevant to the work in this thesis. In addition, the choice of the tools and methodology focused on has also been explained for the proposed work in this work.

Open challenges

Below listed research challenges shall provide enough relevance to the basis of our hypothesis.

- ***SDN – compared with the traditional networks and its monitoring*** Aims at knowing the overview of the state-of-art of SDN and differentiate the networking into the control and data plane. Comparing it with the current networks can be analyzed for a broader perspective of the security features and possibilities of monitoring.
- ***Analyzing new SDN vulnerabilities*** Aims at analyzing the possible features of the SDN that are vulnerable to attacks. Initially, the focus is on the open-source solutions, i.e., Open vSwitch which provides the vulnerabilities that can be exploited in the data and control plane devices.
- ***Integrating SDN with the Pulsar Framework*** Aims at notifying the neighboring networks about the ongoing attacks and pass on relevant details to them. The SDN after monitoring and detecting the traffic to be malicious or legitimate may have the capability to publish to the various users connected to the network. These features of publishing and subscribing of the events and other security features in Apache Pulsar shall be analyzed.
- ***Optimally mitigating these attacks:*** The objective is to find a framework which can intelligently provide mitigation to the DDoS attack in the SDN environment by optimally tracing-back the attack source and maintaining the central knowledge of the network, passing on to other networks.
- ***Load Testing the framework*** Aim is to improve the performance of the framework proposed by testing it in a network with a higher amount of load. As a result, the proposed model would be more scalable and would be resilient to the attacks.

Research Tasks

In order to achieve the goal of this research as discussed above, the following steps need to be performed:

- To learn the behavior, characteristic features and the differences between a normal traffic and abrupt traffic to know the behavior of a DDoS attack by building a real-life physical environment to run such kind of an attack.
- Analyzing the traffic and extracting its characteristic features in order to know the behavior of a DDoS attack traffic.
- Arrange these patterns and distinguish these in order to collect it in a detection collector.

Summary

This chapter basically focused on the main objective and goals that we want to achieve from this work. The author has listed the various tasks that shall be carried out during this research. A method of measuring

ISSN No:-2456-2165

the bandwidth usage and the packet speed/density has been evaluated in order to distinguish the traffic to be genuine or malicious. In addition, this method of detection is performed at both the switch and the host level. The IP addresses/hosts that are identified as malicious are stored as a part of the detection mechanism, and published to the Pulsar topic. Following that, other neighboring networks would be subscribing to this Pulsar topic and eventually shall block the same host IPs. This exemplifies a collaborative model of detecting the edge-based attack detection and mitigation approach.

ISSN No:-2456-2165

CHAPTER 4
DESIGN OF THE ARCHITECTURE

Detection and mitigation of DDoS attacks is a typical problem and in case of the cloud environment, it becomes a bigger challenge. A cloud network, however, cannot be totally isolated from the traditional networks. There is no doubt that the data centre networks are more complex in their architecture but their foundation is laid on the traditional network architecture. And, in turn, these networks are becoming to be more robust and scalable in the real world.

There have been various approaches that have been suggested in the context of the network in transition which depicts all the specification, open questions, and challenges in building defence modules against DDoS attacks in a cloud architecture. The expanding nature of the mitigation mechanisms along with the evolution of the Internet can be depicted in the approaches that have been discussed.

Proposed Architectural Model

There have been new promising approaches with the advancement of the networks in the context of the cloud environment. These include Software Defined Networking (SDN) based ideas integrated with the collaborative approach of using Apache Pulsar to communicate with other neighboring networks. On analyzing the various existing approaches and their features such as the level of operation, time to respond, time to cooperate with other devices, the author has divided the active response into two main categories as below: -

- Proactive approach in which the steps were taken to monitor and control the potential impact of the attack before it happens.
- A reactive approach, which identifies the abrupt traffic behavior and notifies the Pulsar to communicate with the neighboring networks in real time.

Several terms and methods that have been analyzed and used in this work need to be familiarized to get a background information about them, their architecture and their techniques.

The system model is based on a testbed using Mininet Network emulation software. The reason for choosing such software is because it uses Linux containers and Open vSwitch to allow realistic virtual networks of hosts and switches to be constructed using a virtual machine.

Secondly, we have the Floodlight OpenFlow Controller whose default behavior offers elementary connectivity, and so it was selected for the testbed. As a result, simple performance optimizing applications can be developed since they do not require to be concerned with sustaining connectivity and also it is open to emphasize on implementing optimization.

In addition, Apache Pulsar has been used as a pub-sub messaging system. This is because it can publish and subscribe the messages to collaborate by communicating with other neighboring networks.

Choice of Tools for this Proposed Mechanism

A short technical overview of SDN and NFV has been provided above in the section of related works in Chapter 2. In this sub-section, these principles have been further explored in order to justify the selection of architecture and tools that shall be integrated into this framework. The justification for these principles and tools has been listed below:

- **NFV** can provide high-performance packet processing and run network functions on the commodity servers efficiently [7]
- **SDN** as an architecture can control network behavior centrally and steer flows in and across data centres

flexibly [4]. SDN provides a programmable network platform to encourage innovation for the detection or mitigation of ongoing cyber-attacks.

SDN provides quite a good separation among virtual networks which would help one to experiment, deploy and implement new ideas as like in a real environment.

- **Floodlight** offers a module loading system that makes it simple to extend and enhance. It is also easy to set up with minimal dependencies. In addition, it supports a wide range of virtual and physical OpenFlow switches [42].

Floodlight is designed to be highly performant. The core architecture of Floodlight is multithreaded.

- **Apache Pulsar** is used as a pub-sub messaging system due to its design that provides low publish latency and persistent storage that guarantees message delivery to the intended target. In addition, Pulsar can be seamlessly expanded to hundreds of nodes demonstrating its horizontally scalable nature [43].
- **Mininet** is chosen as the network emulator to create the testbed since it is quickly reconfigurable and restartable. It also scales easily and provides good bandwidth which would be helpful for the evaluation of the proposed framework. In addition, it comes with an out of the box command line interface (CLI) which helps in running network-wide tests [44].

Summary

In this chapter, the author has focussed on the proposed solution and the architecture that has been implemented in this thesis. Here, a pub-sub messaging system has been introduced which communicates with other neighbouring networks in order to defend against the attack collaboratively. In addition, even the mechanism of detecting a traffic to be malicious and blocking them in Floodlight has also been discussed. In order to produce a realistic output, different SDN environments with Mininet topologies integrated into it are used as simulators. Then it is aimed to verify and evaluate the contribution by comparing it against other approaches. The assessment of this research is based on the accuracy, scalability, and performance compared with the results of some other approaches. Further, in the remaining chapters, the aims and the open challenges are tackled, designed, experimented and implemented and tested for making it as an effective solution.

ISSN No:-2456-2165

CHAPTER 5
ENVIRONMENTS AND EXPERIMENTS

Introduction

In this chapter, the environments used to execute different DDoS experiments have been discussed for the purpose of research, whereby we identify and execute various experiments that simplify the design process that is described in the following chapter. These experiments also guide in identifying the characteristic features that can differentiate malicious packets from the legitimate ones.

The detection process is one of the most vital components in this work and failure in distinguishing the malicious packets from the legitimate ones can have serious consequences. Hence, the environments and technologies involved in the below experiments and technologies must be validated conscientiously.

There would be two steps to this model, with a proactive and reactive mechanism. The first step begins by exploring the process of retrieving the packets from the networks in different physical environments where each set of packets is organized and prepared for identifying its features to differentiate the legitimate ones from the one being malicious. This process included scanning of all the ports or the hosts in the network with different pings from different addresses and launching a DDoS attack in the network. The results of the various IP addresses sending packets in the network are collected, carefully analysed and compared with the genuine traffic at a threshold level in order to verify the features and distinguish the genuine traffic from the illegitimate one. This part of the process required an intensive understanding of the SDN Architecture, the travel of packets captured in flow tables and their communication with each other in the network. Any packets identified to have a different level of traffic flow in the network or trying to overwhelm the network are blocked and stored in the collector as part of the detection mechanism. The second step is the collaboration phase whereby, the IP addresses collected on the collector publishes it to the pub-sub messaging system, called the Apache Pulsar, in order to subscribe it to the neighbouring networks. As a result, the other networks in the neighbouring domains would be aware of the attacker to block them.

Environment Setups

In this section, the process of building the virtual environment has been presented to identify the characteristics features of both normal and malicious traffic.

The purpose of this kind of an environment is to launch the DDoS attacks with integrated SDN and Apache Pulsar as described in the system architecture in Chapter 4. This environment is chosen because the risk of packets flooding out from the virtual environment is minimal as it is more controllable from a configuration perspective. Moreover, it is not that expensive to configure devices or launch DDoS attacks in this environment, whereas the same process in a physical environment would take days for deployment followed by a check on all domains in the organisation.

The virtual environment is an isolated environment and such isolation protects the physical domain network from accidental DDoS attacks ensuring both scalability and security enabling this work to be undertaken in a safe manner. The virtual environment that has been used in this work is built using Oracle VirtualBox (VBox) [41] where clones of physical servers, networks, routers, and switches were created and duplicated to introduce quite a similar physical network.

Mininet is used to define a topology consisting of various switches, hosts and a centralised controller, which is the Floodlight controller in the network.

The virtual environment that is chosen runs on the following configurations:

- A physical Lenovo machine with 4 processors, 6GB RAM, 2 CPU Cores.
- Oracle Virtual Box Application (VBox).

Ubuntu has been selected to be the operating system due to its security, performance, and compatibility with VBox. The installation and all the configuration have been done through a graphical user interface (Oracle VM VirtualBox Manager).

Such a graphical user interface helps the user to choose various distinct network settings using different virtual devices. This also helps users to combine wired and wireless networks with network capacity according to the user's choice or requirements. It even guides the interface to choose the storage type of data or can even change the physical Media Access Control (MAC) address if required.

Tools and Analysis

The attack tools that have been used in the virtual isolated test-bed environment are carefully used only for the part of this research and learning, and only to identify or understand the characteristics features of the genuine and malicious traffic for the purpose explained in Section 4.2.

The initial foundation of this framework began by selecting the attack methodologies and tools that are generally used by attackers to launch such kind of attacks in the network. There are various types of DDoS attack tools, some of which are weak and some effective but to be effective, the attack must overwhelm the network to introduce network latency or crash the network of the victim host. And therefore, the focus in this experiment was on tools that are most effectively used and are productively meaningful for our purposes. There are a lot of tools that yield better results than others, although the same methodologies and protocols are used due to their implementation and architectural design.

Testbed Layout

The testbed has been designed mainly by two components:

- Pulsar – The Pub-Sub messaging system
- Individual Standalone Networks – serving as neighbouring networks polling a common Pulsar topic for information.

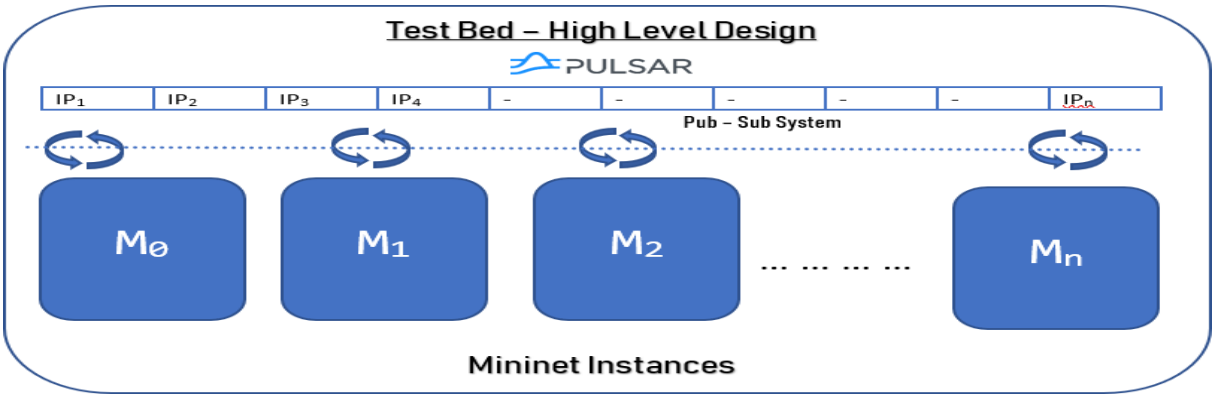

Figure 1 - High-Level Design

ISSN No:-2456-2165

Pulsar

Apache Pulsar is taken to be the Pub-Sub messaging system of choice for this research. For this experimentation, the Docker image for the 2.0.1-incubating version has been downloaded which can be shown in *Figure-2* below.

```
sheetaldash@sheetaldash-VirtualBox:~$ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
apachepulsar/pulsar   2.0.1-incubating    d666846e3e04        2 months ago        1.06GB
```

Figure 2 - Docker Images

Pulsar provides both standalone and clustered mode. For this research, Pulsar container is run in standalone mode. This is helpful because it encapsulates all the components and runs in a single Java Virtual Machine (JVM) process. In addition, this mode runs in a single machine and is quite helpful in case of the development of such a framework.

For this experiment, two different ports have been used on Pulsar. One is the port 6650 for the Pulsar broker and the other being 7080, opened for Pulsar Restful API. These have been represented in the *Figure-3* below.

```
sheetaldash@sheetaldash-VirtualBox:~$ docker run -it \
> -p 6650:6650 -p 7080:7080 \
> -v /home/sheetaldash/data:/pulsar/data \
> apachepulsar/pulsar:2.0.1-incubating \
> bin/pulsar standalone --wipe-data
```

Figure 3 - Invoke Docker Container to start Pulsar

Further, as we start the Pulsar, the client starts with the configurations that have been depicted in the *Figure-4* for the purpose of this research.

```
21:29:31.901 [main] INFO  org.apache.pulsar.client.impl.ConsumerStatsRecorderImpl - Pulsar client config: {
  "serviceUrl" : "pulsar://127.0.0.1:6650",
  "operationTimeoutMs" : 30000,
  "statsIntervalSeconds" : 60,
  "numIoThreads" : 1,
  "numListenerThreads" : 1,
  "connectionsPerBroker" : 1,
  "useTcpNoDelay" : true,
  "useTls" : false,
  "tlsTrustCertsFilePath" : "",
  "tlsAllowInsecureConnection" : false,
  "tlsHostnameVerificationEnable" : false,
  "concurrentLookupRequest" : 5000,
  "maxLookupRequest" : 50000,
  "maxNumberOfRejectedRequestPerConnection" : 50,
  "keepAliveIntervalSeconds" : 30
}
```

Figure 4 - Pulsar Client Config

Now that we have started the pulsar and the container is started, it would show up as a Docker process. If this needs to be identified, the following commands can be used and results, as shown in the *Figure-5*, would clearly represent the details of the container.

In this figure, the ***apachepulsar/pulsar*** that has been highlighted is the image that is used and ***2.0.1-incubating*** represents to be the tag of the image.

```
sheetaldash@sheetaldash-VirtualBox:~$ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED
88895f80c8bb        apachepulsar/pulsar:2.0.1-incubating   "bin/pulsar standalo…"   9 minutes ago
sheetaldash@sheetaldash-VirtualBox:~$
```

Figure 5 - Docker Image of Pulsar

Floodlight Controllers

For the purpose of this research and its experimentation, two floodlight controllers have been used for two SDNs. The reason for using this is to represent the collaborative approach mechanism for the proposed design in this work. The Floodlight controller being an integral part of SDN uses two different ports, one being the 6653 and the other being the 16653. Similarly, for accessing the Rest API of this floodlight controller, their corresponding ports are 8080 and 18080 respectively.

The *Figure-6* below represents the names of the two floodlight installations with different ports as explained above.

```
sheetaldash@sheetaldash-VirtualBox: /opt
File  Edit  View  Search  Terminal  Help
sheetaldash@sheetaldash-VirtualBox:~$ cd /opt
sheetaldash@sheetaldash-VirtualBox:/opt$ ls -ld floodlight*
drwxr-xr-x 11 sheetaldash sheetaldash 4096 Jul  3 23:35 floodlight
drwxr-xr-x 11 sheetaldash sheetaldash 4096 Jul 15 17:42 floodlight_nw1
```

Figure 6 - Floodlight Installations

Mininet Networks/Topology

Each of the Mininet instances that have been used for this dissertation can have primarily three kinds of switches *(Figure-7)*:

- One, being the Source Switch (es), which shall be multiple in numbers, used by various hosts (being used as potential Internet of Things (IoT) based devices). Out of the various hosts, there will be few sending legitimate traffic and a few malicious ones.
- Second being the Aggregation Switch which is the switch that connects all the source switches to the destination.
- And finally, the Gateway Switch which acts as a deep packet inspection (DPI) switch that drops packets based on whether the traffic is deemed to be malicious or not.

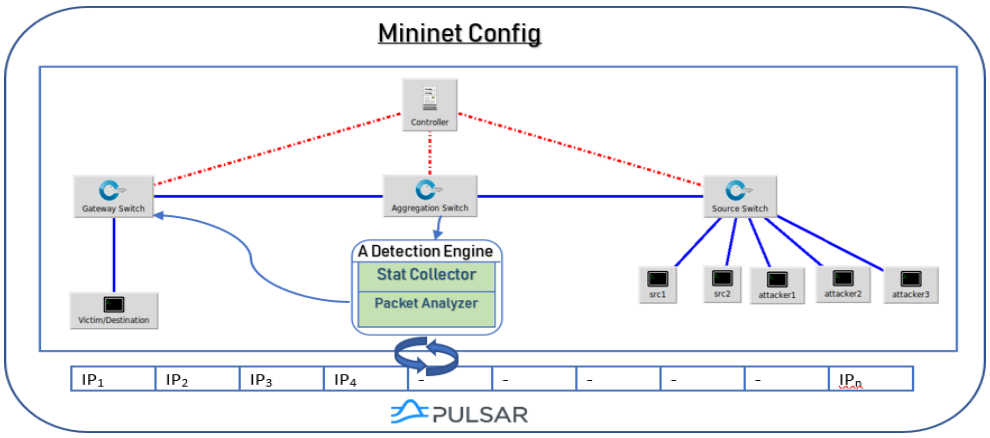

Figure 7 - SDN Architecture

The *Figure-8* below represents a sample Mininet Topology Command line interface that shows that the Mininet topology has 10 hosts (h), 3 switches(s) and a Floodlight controller (c).

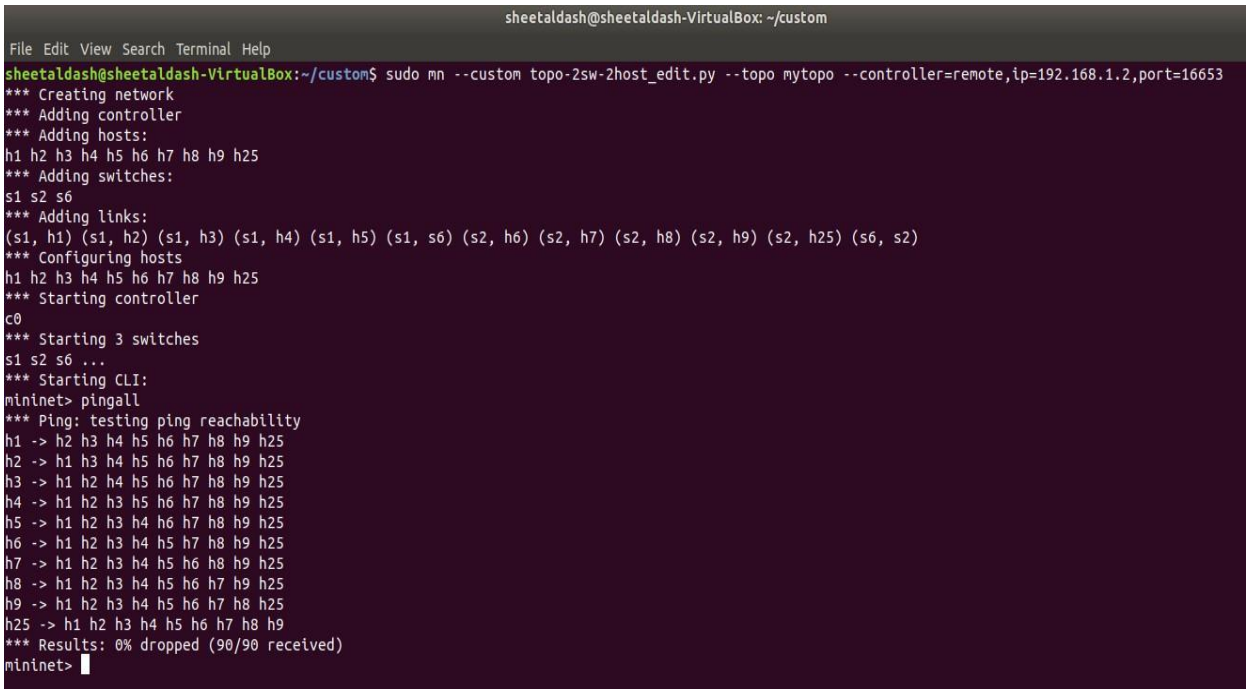

Figure 8 - Sample Mininet Command Line Interface

All these switches have been connected to a central controller which is the Floodlight controller that would be maintaining an Access Control List (ACL).

The graphical user Interface (GUI) of the Floodlight controller has been shown in the *Figure-9* below including the topology representation that has been created above in Mininet.

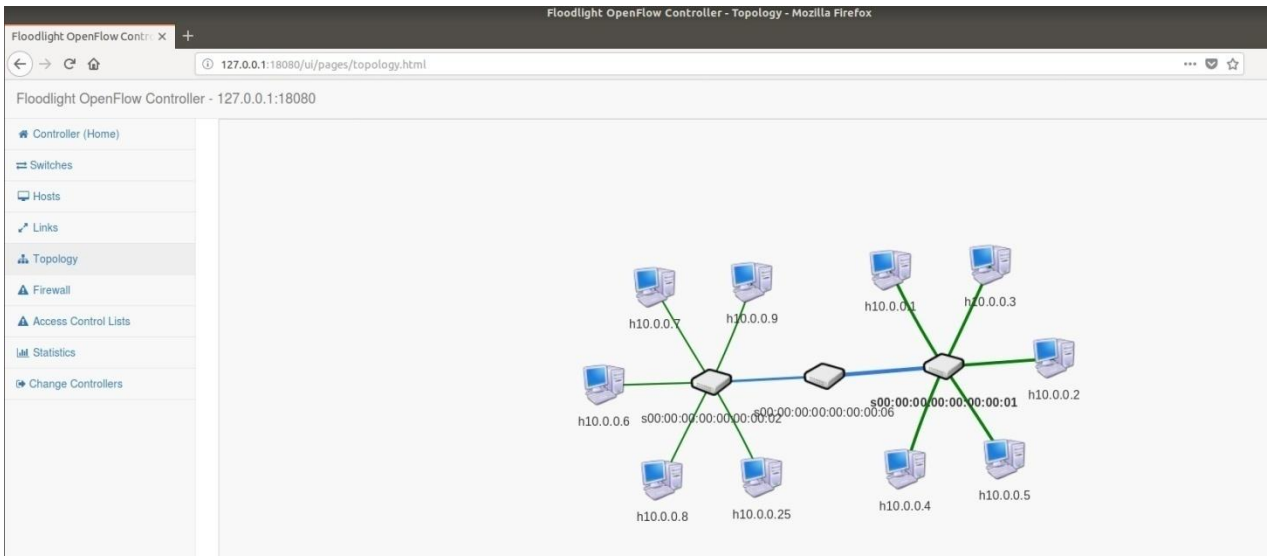

Figure 9 - Sample Mininet Topology

In order to show a new network interface prefixed with the names of the switch (being s1, s2, s6) for each of the port that has been opened by the Mininet, *ifconfig* has been used which can be depicted in the screen capture below *(Figure-10)*.

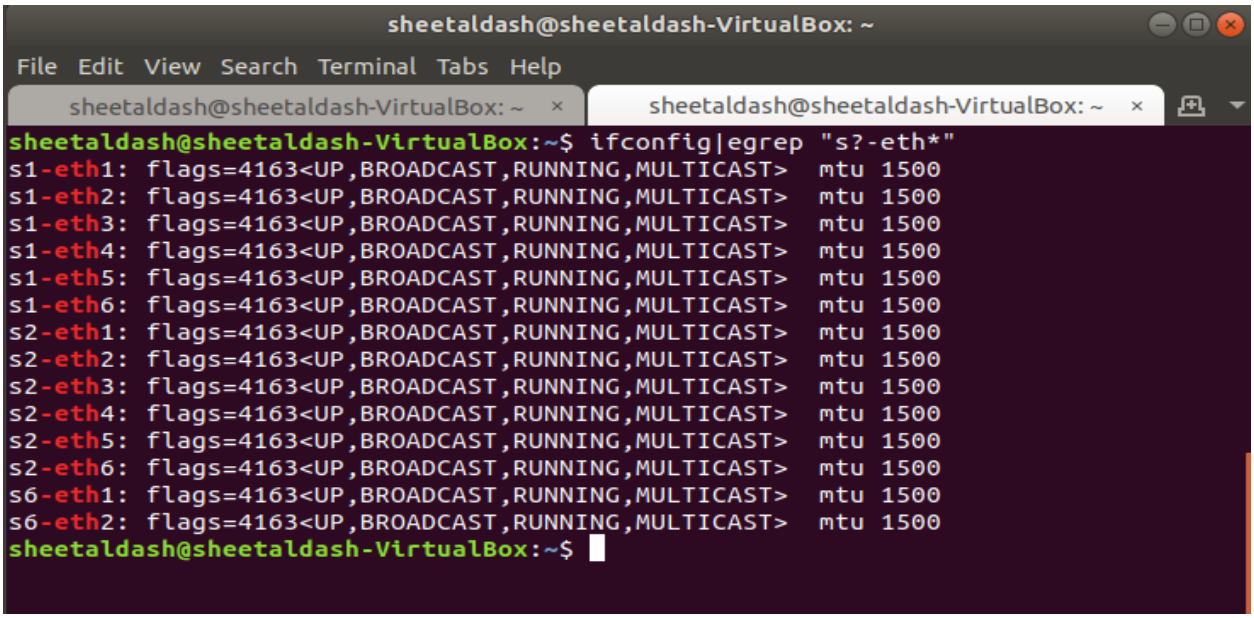

Figure 10 - Network Interfaces opened up by Mininet

Mininet, by default, sends the network traffic as Internet Protocol version 6 (IPv6). This means that the output of the Rest API that is being represented is of the nature of the IPv6 addresses. In the experiment that has been used in this thesis, the ACL module of the Project Floodlight has been used in order to block the host IPs. For the purpose of blocking, the addresses that can be used is the Internet Protocol version 4 (IPv4).

In order to enable the IPv4 address and override the default behaviour, the custom topology script can be modified on the host creation step. This modification can be done by using the below command (*Figure 11*), in the custom Mininet topology creation python script (as provided in *Appendix-1*).

```
self.addHost('h1', ip='10.0.0.1', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
```

Figure 11 - Add Host / Enable IPv4

Traffic Monitoring

The traffic across the network is being monitored for the below parameters:

- Bandwidth Usage
- Packet Speed/Throughput
- IP Density

Bandwidth Usage

The bandwidth usage can be captured from the floodlight's captured statistics. And therefore, in order to collect the statistics, this needs to be enabled first since the stat collection in Floodlight is not enabled by default.

The script *check_bw_final.py* [45] is the one which helps in enabling the statistics. The *Figure-12* below represents the script which helps in enabling the statistics.

cids_using_pulsar / check_bw_final.py

```
105    def stat_enable():
106
107        try:
108            stat = requests.put("http://127.0.0.1:18080/wm/statistics/config/enable/json")
109            print "Enabled statistics..."
110
111        except Exception as e:
112            print"error while enabling statistics :" , e
113
```

Figure 12 - Enable Statistics

Floodlight controller provides bandwidth usage statistics at a "per switch – per port" granularity via a GET call which gives the below output:

[{
u'updated': u'Sun Sep 02 21:25:36 BST 2018', u'bits-per-second-tx': u'0',
u'link-speed-bits-per-second': u'10000000', u'dpid': u'00:00:00:00:00:00:00:0b',
u'bits-per-second-rx': u'0', u'port': u'6'
}]

In order to capture the usage for all the ports for all the switches, the method *topology_info*() in this script, first finds out all the switches & hosts on the network. This is clearly represented in the section of the method that has been shown in the *Figure-13* below.

cids_using_pulsar / check_bw_final.py

```
8     def topology_info():
9         print('\n<----------------------------SUMMARY------------------------------>')
10        data1 = requests.get("http://127.0.0.10:18080/wm/core/controller/summary/json")
11        dat1 = data1.json()
12        number_of_switches = dat1 ["# Switches"]
13        number_of_hosts = dat1 ["# hosts"]
14        print '# Switches Connected: ', number_of_switches
15        print '# Hosts Connected: ', number_of_hosts
16        print '------------------------------------------------------------------\n'
17        return( number_of_switches , number_of_hosts )
```

Figure 13 - Get Topology Info

This script of *check_bw_final.py* then iterates for all the switches. It scans all the ports on these switches in order to capture the bandwidth usage on each of them. This usage is captured in bits per second as highlighted in the *Figure-14* depicted below.

```
84              a = requests.get("http://127.0.0.1:18080/wm/statistics/bandwidth/"+DPID+"/"+str(port)+"/json")
85              b = a.json()
86              bandwidth = int (b[0]["bits-per-second-rx"])
87              dpid=str(b[0]["dpid"])
88              ipv4=find_host(dpid, str(b[0]["port"]))
89              print "\tport :", b[0]["port"]
90              print "\t\tBits per Second :", bandwidth
```

Figure 14 - Bandwidth Usage from Floodlight

Packet Speed

Traffic Sniffer

The speed of the packet would be ascertained by the number of packets that have been sent or received within a stipulated period of time which is regarded the same as the Polling interval that has been defined in Section 5.4 in this thesis.

In order to monitor the traffic, all the network interfaces created as a part of the Mininet topology would be monitored. This monitoring is done by the use of the tool, known as Scapy in Python. This is quite a powerful tool which helps to scan these interfaces and even has the ability to decode packets of various protocols, send them on the wire and capture them. The script *sniff- traffic.py* has the main method which initiates an asynchronous process for each of the network interfaces in the topology. This has been represented in the code snippet shown below in the *Figure- 15*.

cids_using_pulsar / sniff-traffic.py Edit

```
39          producer = client.create_producer(SNIFFER_TOPIC, max_pending_messages=100000)
40          sniff(filter="ip",iface=switch,prn=sniffPacketsWrapper(producer))
41
42      def main():
43
44          print "custom packet sniffer"
45          switches=commands.getoutput('ifconfig|egrep "s?-eth*"|grep -v s6|awk -F":" \'{print $1}\'')
46          for switch in switches.splitlines():
47              if switch != '0':
48                  p=Process(target=sniffPacketsSwitch, args=(switch,))
49                  p.start()
50      if __name__ == '__main__':
51          main()
```

Figure 15 - Initiate Asynchronous Sniffers

In the other section of the script, each process uses the sniff function of scapy that helps in filtering the IP traffic from the network which has been shown in the code snippet below *(Figure-16)*.

cids_using_pulsar / sniff-traffic.py

```
36
37      def sniffPacketsSwitch(switch):
38          # Build a producer instance on a specific topic
39          producer = client.create_producer(SNIFFER_TOPIC, max_pending_messages=100000)
40          sniff(filter="ip",iface=switch,prn=sniffPacketsWrapper(producer))
```

Figure 16 - Filter IP Traffic

ISSN No:-2456-2165

While sniffing the traffic, the response packets are being filtered out. This is because when we send a request to a host, an acknowledgment from it would come as a response to that request which would cause someone to interpret wrongly that both the hosts are the source.

In other words, if the IP traffic has the layer as Internet Control Message Protocol (ICMP), the response traffic (that is the 'echo reply') would have the *type attribute* as 0 and the request traffic (that is the 'echo request') would have the *type attribute* to be 8. And, only the records with *type attribute* '8' are being captured in this experiment in order to avoid duplicating the counts.

As can be seen in the below *Figure-17*, one packet is sent from source IP being 10.0.1.2(Node h22) to the destination IP i.e. 10.0.1.4. The command used for this purpose was ping.

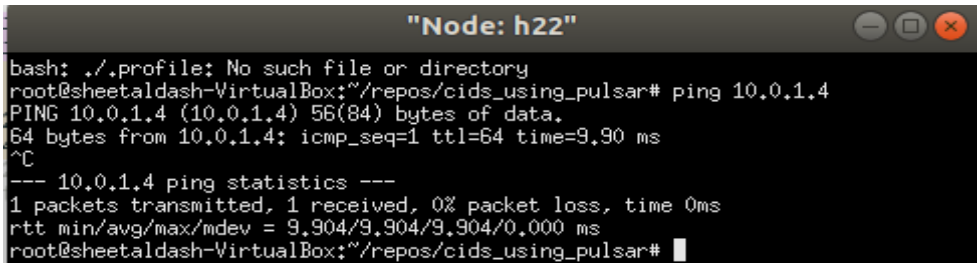

Figure 17 - PING Test

The packet when analysed shows two entries in Wireshark. The info of the packet reveals that the packet #3 in the below *Figure-18* is the Ping Request and the packet #4 is the Ping Reply.

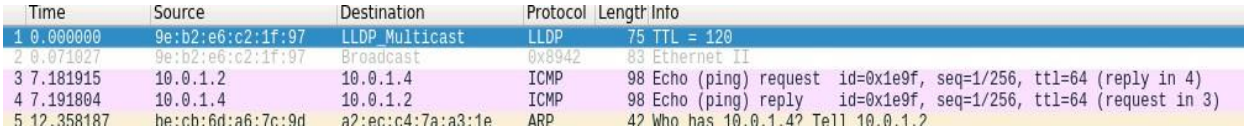

Figure 18 - Wireshark Capture of Ping Test

The packet #3 shows the type attribute as 8 and the packet #4 has the type attribute as 0, as can be seen, highlighted in the *Figure-19* below.

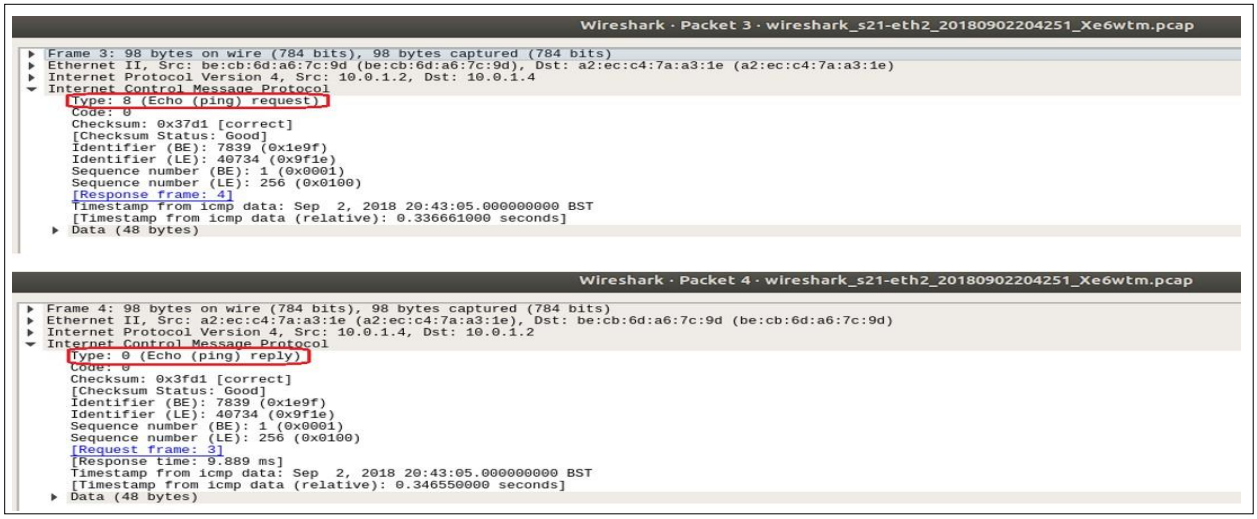

Figure 19 - Deep Packet Inspection of Wireshark Captures

As discussed earlier, for a network traffic that doesn't have an ICMP layer, the type attribute does not exist. Therefore, ping request and reply need to be identified using different condition. The below *Figure-20* shows a couple of packets sent from source IP 10.0.1.2 to destination IP 10.0.1.4 using *hping3*.

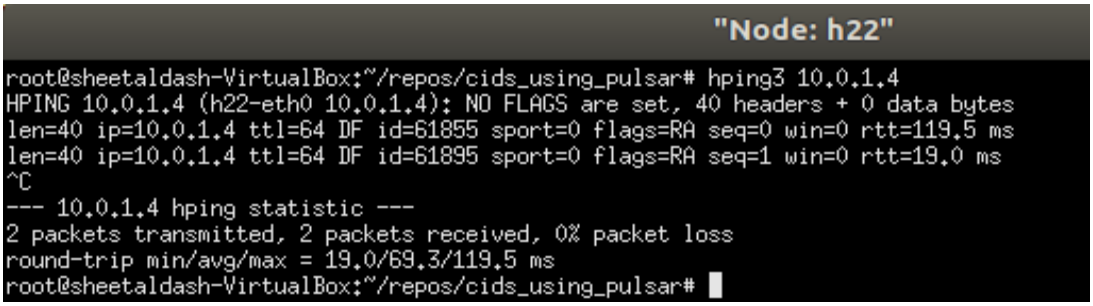

Figure 20 - HPING3 Test

The deep packet inspection (DPI) of one of the packets shows a TCP layer instead of an ICMP layer. As can be seen in the below *Figure-21*, the packet #1 which is the Ping Request has *flags* set as 0 and the Ping Reply has *flags* set as a non-zero value.

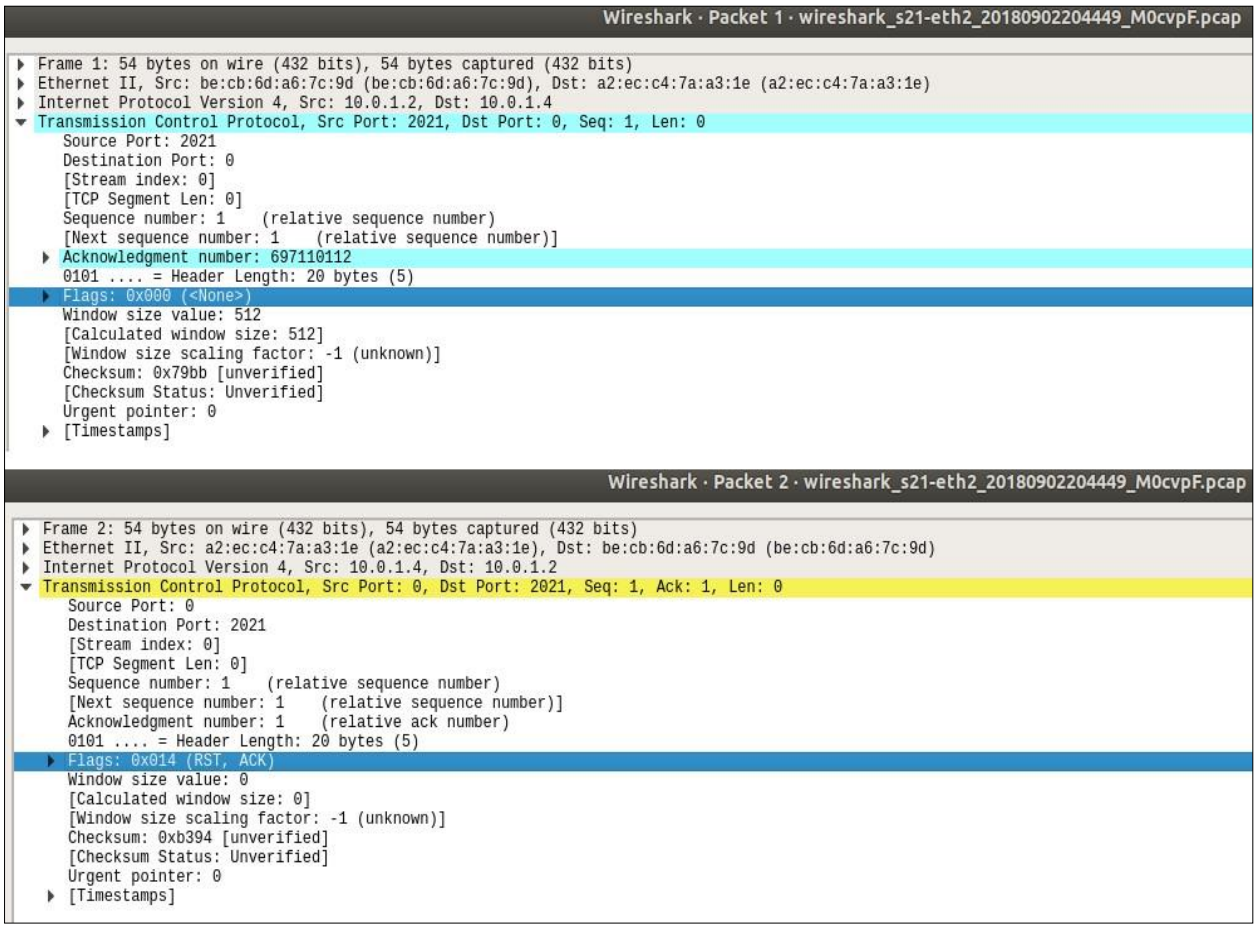

Figure 21 - Deep Packet Inspection of HPING3 Captures

The sniffer program, therefore, checks the packet to see if it has an ICMP layer. If it has an ICMP layer, the code is shown below (*Figure-22*) looks for the *type attribute* and filters out any packets that do not have the *type attribute* as '8'. And, if the packet does not have an ICMP layer, the code looks for the flags and filters out any non-zero flags, which are the 'Ping Reply' packets.

cids_using_pulsar / sniff-traffic.py

```
15    def sniffPacketsWrapper(producer):
16        def sniffPackets(packet):  # custom custom packet sniffer action method
17            if not (packet.haslayer(ICMP)):
18                if packet[IP].flags == 0:
19                    pckt_src = packet[IP].src
20                    pckt_dst = packet[IP].dst
21                    pckt_ttl = packet[IP].ttl
22                    print
23                    "IP Packet: %s is going to %s and has ttl value %s" % (pckt_src, pckt_dst, pckt_ttl)
24                    producer.send_async(pckt_src + "|" + pckt_dst, callBack)
25                    return
26            elif packet.haslayer(ICMP) and packet[ICMP].type == 8:
27                pckt_src = packet[IP].src
28                pckt_dst = packet[IP].dst
29                pckt_ttl = packet[IP].ttl
30                print
31                "IP Packet: %s is going to %s and has ttl value %s" % (pckt_src, pckt_dst, pckt_ttl)
32                # logging.info('Source|Target: %s ', pckt_src + "|" + pckt_dst)
33                producer.send_async(pckt_src + "|" + pckt_dst, callBack)
34                return
35        return sniffPackets
```

Figure 22 - Ignore 'PING REPLY' Packets

Now, after the traffic has been sniffed, the source and destination IPs are distinguished and is written to the Pulsar topic in asynchronous mode. Here, the mode has been chosen to be the asynchronous mode for maximum performance. This mode would put the message on the queue and return without waiting for an acknowledgement. The risk in this kind of a mode is the queue could grow in size if many messages are not acknowledged. And therefore, the maximum pending messages *(Figure-16)* is an attribute or parameter that is set at the time of creating the producer.

The format of the message that is written to the Pulsar topic is the packet source IP followed by the packet destination IP, separated by a pipe delimiter.

Collector & Loader

Considering the database operations to be usually resource intensive and therefore could be detrimental to performance, the sniffed traffic is not loaded directly to the database. Rather, the messages are written to the Pulsar topic by the traffic sniffer function. And thereby, they would be picked up after every n^{th} message and loaded to a table, named *ip_sniffer* in the database for the analytics job to run on it which would be explained below in the thesis. Consequently, this would, in turn, reduce the number of hits to the database by *1/n* and also, the performance level is maximized due to the highly scalable nature of Pulsar in response to high-speed network traffic.

Further, the load frequency (n) should not be too low as that would trigger very frequent loads to the database, which would have a detrimental impact on the performance. The load frequency should not be too high either, because that would mean the data would not be moved from the pulsar topic to the tables for analysis till the required number of packets for a load is received. This would even reduce the accuracy of the analysis as many packets would not be considered.

The below code snipper *(Figure-23)* shows the counter that goes up to the load frequency. Before the counter has reached the count, the messages keep getting appended to the list and once it reaches the count, it gets loaded to the *"ip_sniffer"* table.

cids_using_pulsar / consumer-load.py

```
40        logging.info('Created consumer for the topic %s', SNIFFER_TOPIC)
41        while True:
42            try:
43                # try and receive messages with a timeout of 10 seconds
44                msg = consumer.receive(timeout_millis=TIMEOUT)
45                logging.info("Received message '%s'", msg.data())
46                source_ips.append(msg.data().split("|")[0])
47                dest_ips.append(msg.data().split("|")[1])
48                counter=counter+1
49                if counter >= 5:
50                    traffic=list(zip(source_ips,dest_ips))
51                    source_ips=[]
52                    dest_ips=[]
53                    insert_db(traffic)
54                    counter=0
55                consumer.acknowledge(msg)  # send ack to pulsar for message consumption
56
57
58            except Exception:
59                received = 0
```

Figure 23 - Data Polled & Loaded to Tables to Database

The data received would be split using its delimiter *"|"* and then would be loaded to the *ip_sniffer* table in the database. This is clearly spotted in the code snippet below (*Figure-24*) which shows the record being inserted into the table.

```
22        mycursor = mydb.cursor()
23
24        sql = "INSERT INTO ip_sniffer (source_ip, dest_ip) VALUES (%s, %s)"
25        mycursor.executemany(sql, traffic)
26
27        mydb.commit()
28
29        print(mycursor.rowcount, "record inserted.")
```

Figure 24 - Load to IP Sniffer table

Threshold to Differentiate genuine and Malicious Traffic

Various configuration parameters need to be defined in the code for the framework to be able to detect malicious traffic. These parameters are the following:

1.) ***Bandwidth Consumption Threshold:*** This threshold signifies if the bandwidth consumption i.e. *bits-per-second-rx* for any of the ports on any of the switches exceeds the value set for the threshold, and then the connected host is deemed to be malicious.

2.) ***Polling Interval***: This is the time duration after which the data captured during the traffic monitoring stage would be aggregated and analysed for making decisions. This parameter would define the responsiveness of the application to attacks.

3.) ***Threshold of requests per source***: This is the upper limit that has been configured on the number of packets from a source IP address per polling interval. Setting this parameter to an appropriate value is important. This is because a very low value could result in a high number of false positives for the network that is under threat and all its neighbouring networks. Similarly, a very high value could allow a DDoS attack to take place and therefore would defeat the purpose of this intrusion detection framework.

4.) ***Threshold of requests per destination***: This is the upper limit that has been configured on the number of packets from multiple source IPs to a destination IP address per polling interval. Similar to the "threshold of requests per source", the value needs to be carefully considered and set up in order to prevent high false positives or false negatives.

5.) ***LIMIT of Destination IPs block***: When the number of packets per polling interval for a destination IP exceeds the threshold of the requests per destination, this parameter would define the number of top packet contributors, i.e., the source IPs that would be deemed malicious and eventually blocked in order to mitigate the attack. As a result, this would bring down the number of packets per second at the destination IP.

6.) ***ACL Retention Period***: This is the period of time for which all the IPs that have been added to the ACL rules would be retained on the ACL list in Floodlight. Upon expiry of the retention period, the IPs would be released, thereby removing it from the ACL list.

The below table shows the values that are used for the purpose of this research.

Threshold Name	Value
Bandwidth Consumption Threshold	**20,000**
Polling Interval	**5 secs**
Threshold of requests per source	**100 packets per second**
Threshold of requests per destination	**400 packets per second**
LIMIT of Destination IPs block	**Top 3 contributors**
ACL Retention Period	**15 mins**

Table 1 - Thresholds used in the Framework

Another important factor that needs to be considered while deciding on neighbouring networks (i.e., the networks subscribing to the same Pulsar topic for intelligence about malicious hosts) is, networks with similar capacity in terms of the ability to handle high-speed network traffic should be grouped together. If a network with much higher capacity is in the same neighbourhood as other networks that do not have a comparable capacity, it would result in a lot of false positives for the former, since other networks would have set their thresholds much lower and therefore would be blocking hosts who would not have caused an issue with the network with high capacity.

Traffic Analysis & Attack Mitigation

Based on the results from the traffic monitoring explained above, the data collected would be analysed in order to differentiate between the malicious and genuine traffic. For the purpose of analysis, various hosts are identified to be malicious based on the threshold that has been configured. If the hosts exceed this threshold and overwhelm the network then they are deemed to be the malicious ones. And therefore, the IP of the hosts are added to the ACL rules of the corresponding network in the floodlight denying access and blocking the traffic.

At the same time, these IPs are even sent to a Pulsar topic, which is subscribed to by other neighbouring networks. Consequently, the neighbouring networks even pick the same information from the topic and thereby, block the IP of the host on its corresponding controller as well. This demonstrates the collaborative

mechanism of this proposed model discussed in this thesis.

The implementation has been done using various programs/modules. These are explained in the sections further.

Posting of ACL Rules

The job that would be posting the ACL rules would run on all the individual networks. These ACL rules are published on to the Pulsar topic. Each network would be subscribing to the same Pulsar topic by creating its own subscriber like *sub1* and *sub2* in this experiment (as shown below in *Figure 25*)

Each individual subscriber would read from the Pulsar topic and calls the Floodlight Rest API to post the received IP on to the ACL list. Once the processing is complete, it would send an acknowledgement back triggering Pulsar to send in the next message. In other words, it acts like a conventional queue that exhibits the First-In-First-Out (FIFO) mode of dealing with messages.

Figure 25 - Post ACL Rules Scripts

Further, the expected message on the *attackers_topic* would be an IPv4 address. In addition to the address being loaded to the ACL rules in Floodlight, it would also be loaded to a table in the database called *ACL_Retention.* This table contains the IP address of the host and the timestamp at which it was loaded. As a result, this will help perform housekeeping of the ACL rules. The *Figure-26* below shows the python script which performs the activities explained above.

Figure 26 - Post to ACL & Load to ACL_RETENTION Table

ACL Rules Housekeeping

The idea behind the acl-rules-housekeeping mechanism is to remove the block on an IP address that has been blocked previously after a certain period of time. The purpose of doing this is not to overload the database with loads of data and hamper its performance. In addition, this even allows for course correction in case a host was infected in the past because of its IP being spoofed or have been intruded by someone and is triggering a DDoS attack but it has been clean since then. However, if the host exhibits similar behaviour it

would again be added to the ACL table and blocked.

As can be seen in the below code snippet *(Figure-27)*, this job would look at the *acl_retention* table for any entries older than the value of the parameter "*ACL Retention Period*".

cids_using_pulsar / acl-rules-housekeeping.py

```
20              mycursor = mydb.cursor()
21              mycursor.execute("select blocked_ip from acl_retention where loadtime < now()-interval 15 minute")
22              blocked_ips = mycursor.fetchall()
```

Figure 27 - Fetch IPs blocked for more than 15 mins

Further, in order to clear a specific rule, the API call needs the '*id*' of the entry and is not driven by the IP. Based on the IPv4 addresses retrieved above, the corresponding *id* is fetched from the ACL rules table in floodlight and then the specific rule is cleared. And thereby, deleting the record from the *acl_retention* table (as represented in *Figure-28* below).

cids_using_pulsar / acl-rules-housekeeping.py

```
25              acl_rules = requests.get("http://127.0.0.1:18080/wm/acl/rules/json")
26              acl_rules_data = acl_rules.content
27              acl_rules_json = json.loads(acl_rules_data)
28              for ip in blocked_ips:
29                  for i in range(0, len(acl_rules_json)):
30                      ip=''.join(ip)
31                      if acl_rules_json[i]['nw_src'].replace("/32", "") == ip:
32                          id=acl_rules_json[i]['id']
33                          print("Attempting to remove IP - '" + ip + "' from ACL!!")
34                          command='curl -X DELETE -d \'{"ruleid":"'+str(id)+'"}\' localhost:18080/wm/acl/rules/json'
35                          os.system(command)
36
37                          # Clean up ACL Table
38                          mycursor = mydb.cursor()
39                          sql = "delete from acl_retention where blocked_ip = '"+str(ip)+"'"
40                          mycursor.execute(sql)
41                          mydb.commit()
42                          print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y") + " : Deleted entry from ACL Retention table")
```

Figure 28 - Release Blocks for IPs older than 15 mins

Bandwidth Analysis

The bandwidth monitoring on all the ports of all the switches gives the bandwidth usage, which is then compared with the "*Bandwidth Consumption Threshold*". For hosts that go beyond the threshold (as highlighted in *Figure-29* below), the IPv4 address would be posted to the *ATTACKERS_TOPIC* for it to be blocked.

Once it is blocked, the bandwidth usage would go down approximately to '0' within some time. Meanwhile, in order to avoid multiple duplicate posts to the Pulsar Topic, the program checks if the host is already present on the ACL list. If it is present then the post to the Pulsar topic is skipped otherwise it is posted.

In case of an Aggregation switch or the switch which is not connected to any hosts but only exists as a connector between other switches, the bandwidth could go much higher than the threshold as it would be aggregating the usage from the traffic from all the hosts of the connected switches. Consequently, these ports would not return an IPv4 address and hence, they have been filtered out before posting to the Pulsar topic.

cids_using_pulsar / check_bw_final.py

```
94                if bandwidth > 20000:
95                    if ipv4.replace('\'','') not in acl_json:
96                        print 'DPID: ', dpid
97                        print 'Port: ', str(port)
98                        if ipv4 != "1":
99                            print "Attacker IP : ", ipv4
100                           pulsar_publish(ipv4)
```

Figure 29 - Bandwidth Usage Greater than Threshold

The code snippet as represented in *Figure-30* below shows the method that takes the IP addresses that are believed to be malicious as a parameter in the python script *check_bw_final.py* and writes them to the Pulsar topic for the *post-acl-rules.py* script to read and block them in the network.

cids_using_pulsar / check_bw_final.py

```
59    def pulsar_publish(attacker_ip):
60        ATTACKERS_TOPIC = 'persistent://sample/standalone/ns1/attacker'
61
62        # Setup for basic logging
63        logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.INFO)
64        logging.info('Connecting to Pulsar...')
65
66        # Create a pulsar client instance with reference to the broker
67        client = pulsar.Client('pulsar://localhost:6650')
68
69        # Build a producer instance on a specific topic
70        producer = client.create_producer(ATTACKERS_TOPIC)
71        logging.info('Connected to Pulsar')
72
73        logging.info('Sending ACL Addition Request: %s ', attacker_ip)
74        producer.send(attacker_ip);time.sleep(10)
75
76
77        client.close()
```

Figure 30 - Publish to Pulsar Topic

Evaluation

For the purpose of implementation and evaluation of this work below scripts are used to simulate normal traffic *(Figure-31)* and attack traffic *(Figure-32)*. For the simulation of Normal traffic, 100 packets are sent to multiple targets at the default ping speed. Whereas for an attack traffic simulation, 100 packets are sent at a speed of 500packets per second.

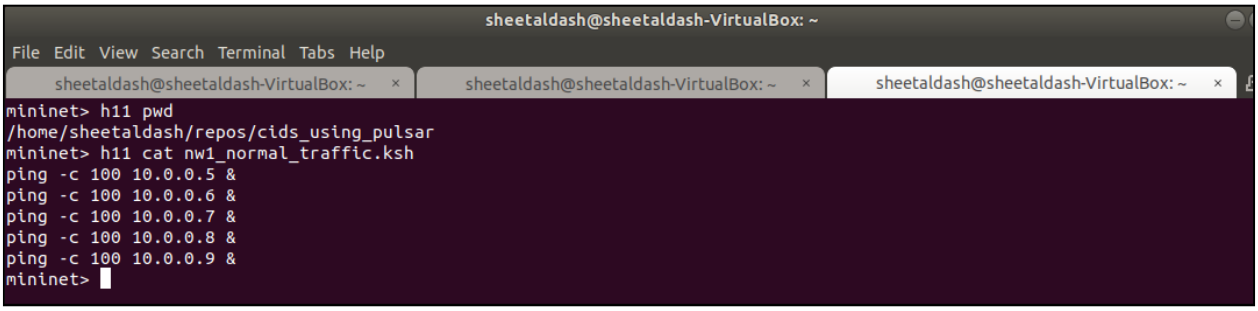

Figure 31 - Normal Traffic Simulator

ISSN No:-2456-2165

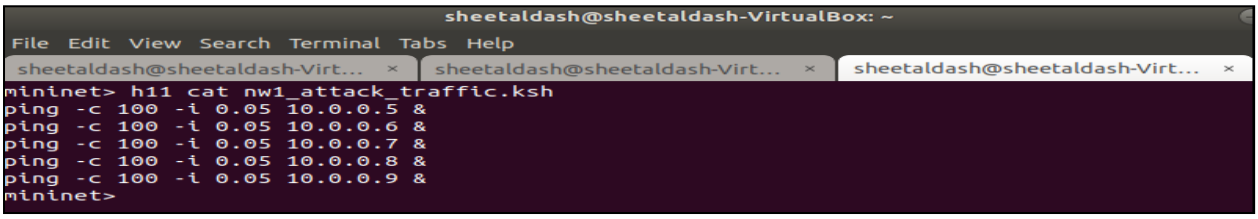

Figure 32 - Attack Traffic Simulator

The *check_bw_final.py* script starts monitoring the bandwidth usage of all the ports of all the switches. As can be seen in the below screenshot, the usage starts from 0 *(Figure-33)* and when the normal traffic simulator script is run, the usage starts to show up on the output *(Figure-34)*.

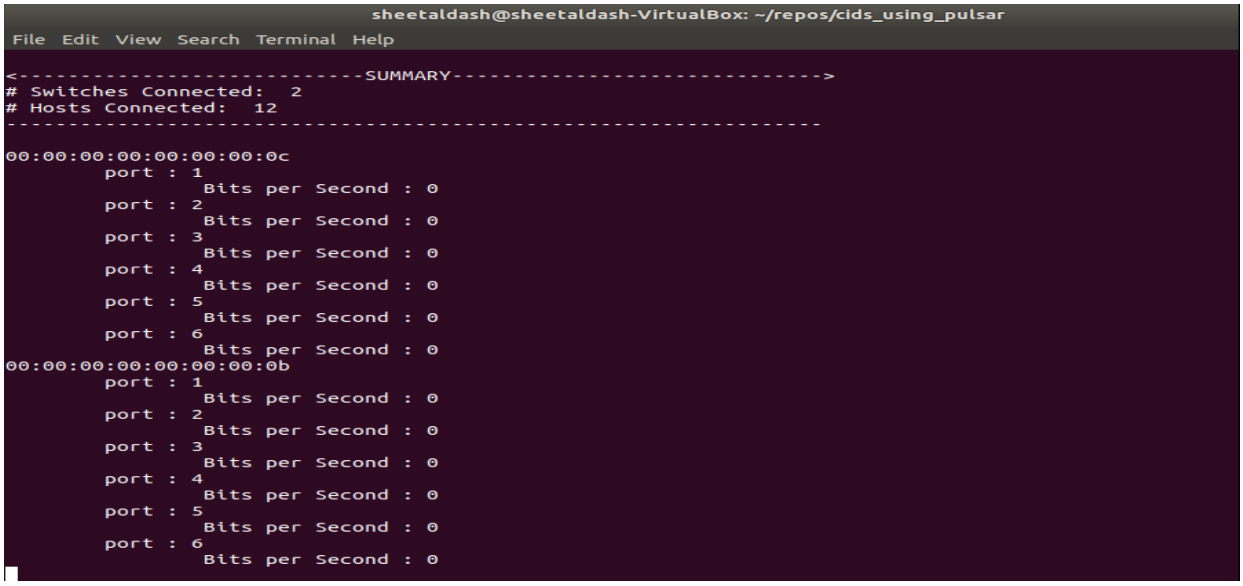

Figure 33 - Initial Bandwidth Usages

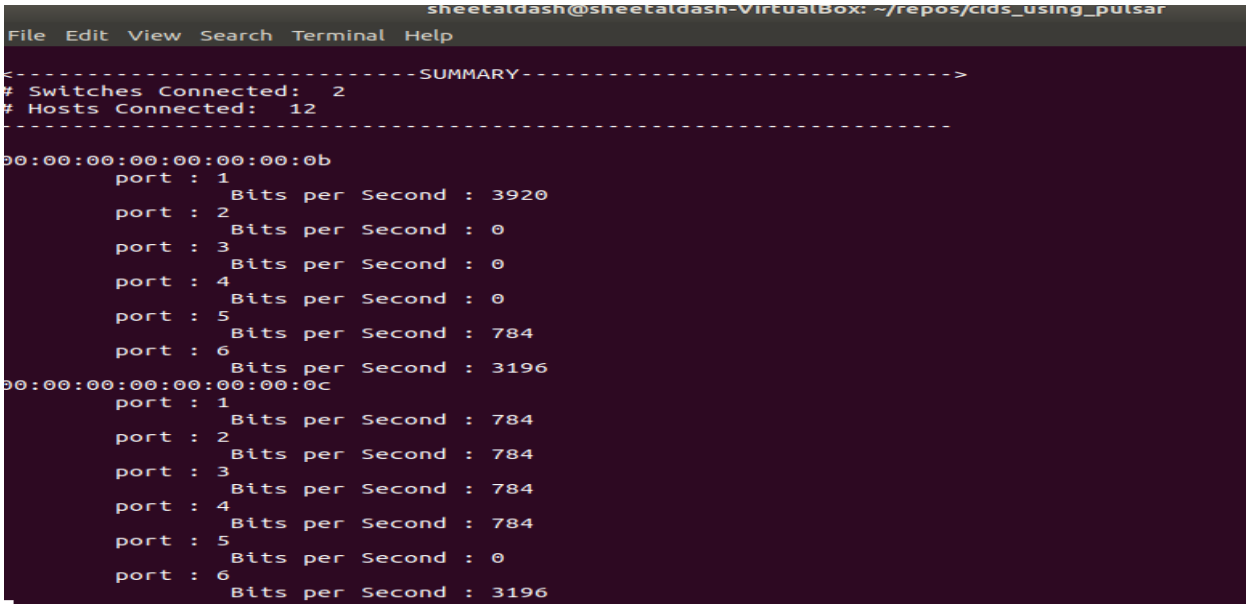

Figure 34 - Bandwidth Usage after starting Normal traffic Simulator

When the attack network simulator script is run, the bandwidth goes beyond the threshold of 20,000 set as per *Table-1*. The host connected to the port is deemed to be malicious and therefore the corresponding IPv4 address is posted to the pulsar topic *(Figure-35)*.

Figure 35 - Attacker IP detected

The below screenshots *(Figure-36 & Figure-37)* show two neighbouring networks picking up the message posted by one of the networks and loading to the ACL in Floodlight and loading to the *acl_retention* table as well.

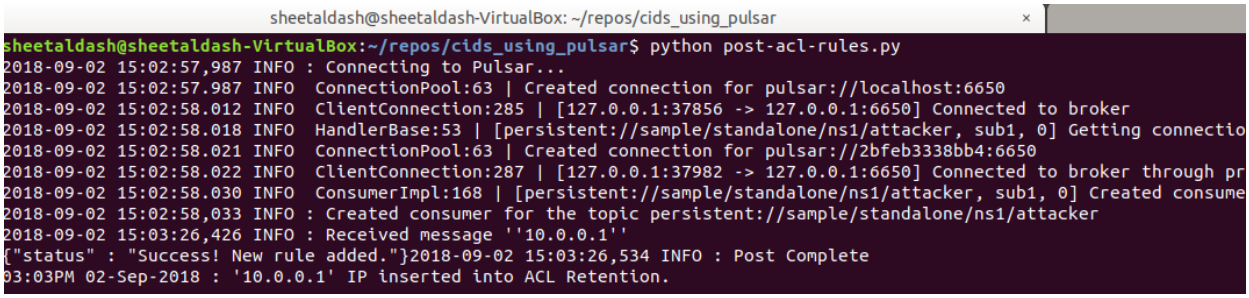

Figure 36 - Attacker IP Blocked on Network 1 i.e. Floodlight - 16653

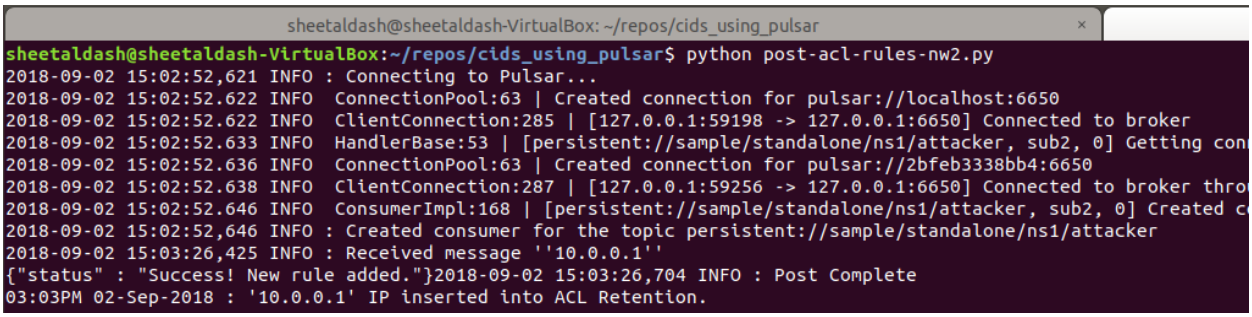

Figure 37 - Attacker IP Blocked on Network 2 i.e. Floodlight - 6653

Each network has its own database i.e. *cids* and *cids_nw2*.

Both databases would have an entry each for the blocked IP address (as in *Figure-38*).

```
mysql> use cids;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from acl_retention;
+------------+---------------------+
| blocked_ip | loadtime            |
+------------+---------------------+
| 10.0.0.1   | 2018-09-02 15:03:26 |
+------------+---------------------+
1 row in set (0.00 sec)

mysql> use cids_nw2 ;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from acl_retention;
+------------+---------------------+
| blocked_ip | loadtime            |
+------------+---------------------+
| 10.0.0.1   | 2018-09-02 15:03:26 |
+------------+---------------------+
1 row in set (0.00 sec)
```

Figure 38 - ACL_RETENTION table loaded on CIDS & CIDS_NW2

Figure-39 below shows the results of an API call to both floodlight controllers. This call aims to get a list of ACL rules on each controller.

```
sheetaldash@sheetaldash-VirtualBox: ~/repos/cids_using_pulsar ×  sheetaldash@sheetaldash-VirtualB
sheetaldash@sheetaldash-VirtualBox:~/repos/cids_using_pulsar$ curl localhost:8080/wm/acl/rules/json
[{"id":1,"nw_src":"10.0.0.1/32","nw_dst":null,"nw_src_prefix":167772161,"nw_src_maskbits":32,"nw_dst_p
ox:~/repos/cids_using_pulsar$
sheetaldash@sheetaldash-VirtualBox:~/repos/cids_using_pulsar$ curl localhost:18080/wm/acl/rules/json
[{"id":1,"nw_src":"10.0.0.1/32","nw_dst":null,"nw_src_prefix":167772161,"nw_src_maskbits":32,"nw_dst_p
ox:~/repos/cids_using_pulsar$
```

Figure 39 - API Call to get ACL rules in each controller

After the retention period i.e., 15 minutes (defined as per *Table-1*) is exhausted, the IP address is released from the ACL rules (*Figure-40)* for both networks.

```
sheetaldash@sheetaldash-VirtualBox: ~/repos/cids_using_... ×  sheetaldash@sheetaldash-VirtualBox: ~/repos/cids_using_... ×  sheetaldash@she
sheetaldash@sheetaldash-VirtualBox:~/repos/cids_using_pulsar$ python acl-rules-housekeeping-nw2.py

Sleeping 5 mins!

Sleeping 5 mins!
Attempting to remove IP - '10.0.0.1' from ACL!!
{"status" : "Success! Rule deleted"}03:18PM 02-Sep-2018 : Deleted entry from ACL Retention table
```

```
sheetaldash@sheetaldash-VirtualBox: ~/repos/cids_using_... ×  sheetaldash@sheetaldash-VirtualBox: ~/repos/cids_using_... ×  sheetaldash@sheeta
sheetaldash@sheetaldash-VirtualBox:~/repos/cids_using_pulsar$ python acl-rules-housekeeping.py

Sleeping 5 mins!

Sleeping 5 mins!

Sleeping 5 mins!
Attempting to remove IP - '10.0.0.1' from ACL!!
{"status" : "Success! Rule deleted"}03:23PM 02-Sep-2018 : Deleted entry from ACL Retention table

Sleeping 5 mins!
```

Figure 40 - After Retention Period block removed

Packet Speed Analysis

For the analysis of the packet speed, the data collected and loaded during the monitoring phase by the Collector & Loader job has been used.

This job's frequency is defined by the "*Polling Interval"* parameter and can be formulated as below,

Equation 1 - Polling Interval Deduction

□□□□□□□ □□□□□□□□ ∝ 1/□□□□□□□□ □□□□□ □□ □□□□□□□

As shown above, the polling interval is inversely proportional to the expected speed of traffic. Alternatively, in networks where the speed is very high, the polling interval should be low or else the DDoS attack would become successful by the time the next iteration of analysis is triggered since timing is very important in case of DDoS attacks.

For the purpose of analysis, two tables have been used which are the following: -

1. **source_sniffer** - This table stores the total number of packets transmitted from each of the source IPs during the polling interval.
2. **dest_sniffer** – is the table that stores the total number of packets received by each destination IP from various source IPs during the polling interval.

The python script *consumer-stat-capture.py* helps in capturing the statistics which initially starts off by truncating the previous analysis results and starts with empty datasets on the above two tables.

Further, the code snippet below *(Figure-41)* conveys that, from the *IP_SNIFFER* table, the number of entries of each source IP (while filtering out source_ip's that are already blocked) is loaded to the *source_sniffer* table. Similarly, a number of entries for each destination IP (while filtering out those entries that have an already blocked IP as its source) is loaded to the *dest_sniffer* table.

```
        # Insert into Source Sniffer
        mycursor = mydb.cursor()
        sql = "insert into source_sniffer(source_ip,count_per_polling_interval,loadtime) (select source_ip,count(1),now() from ip_sniffer \
                where source_ip not in (select blocked_ip from acl_retention) group by source_ip)"
        mycursor.execute(sql)
        mydb.commit()
        print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : " + str(mycursor.rowcount) + " record inserted to Source_Sniffer.")
        mycursor.close()

        # Insert into Destination Sniffer
        mycursor = mydb.cursor()
        sql = "insert into dest_sniffer(dest_ip,count_per_polling_interval,loadtime) (select dest_ip,count(1),now() from ip_sniffer \
                where source_ip not in (select blocked_ip from acl_retention) group by dest_ip)"
        mycursor.execute(sql)
        mydb.commit()
        print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : " + str(mycursor.rowcount) + " record inserted to Dest_Sniffer.")
        mycursor.close()
```

Figure 41 - Aggregate & Load to Source & Dest Sniffer

If the count of any source IP is greater than the "*Threshold of requests per source*", the IP is deemed to be malicious. Similarly, if the count of any destination IP is greater than the "*Threshold of requests per destination*", the top few contributors to that count is selected. The number of contributors to be selected is driven by the "*LIMIT of Destination IPs block*" parameter. These source

IPs are also added to the list of IPs to be blocked *(Figure-42)*.

```
        # Pick Destination Ips with count > threshold
        mycursor = mydb.cursor()
        mycursor.execute("SELECT distinct dest_ip FROM dest_sniffer where count_per_polling_interval>150")
        dest_ips = mycursor.fetchall()
        mycursor.close()

        for ip in dest_ips:
            mycursor = mydb.cursor()
            ip=str(ip).translate(None,"u,'()")
            mycursor.execute("SELECT distinct source_ip FROM ip_sniffer where dest_ip='"+str(ip)+"' group by source_ip order by count(*) desc LIMIT 3")
            ips = mycursor.fetchall()
            source_ips=source_ips+ips
            mycursor.close()
        for ip in source_ips:
            if ip not in block_list:
                block_list.append(ip)
```

Figure 42 - IPs breaching various threshold identified

The *IP_SNIFFER* table is truncated and the list of IPv4 addresses to be blocked is published to the *ATTACKER_TOPIC* (as shown in the *Figure-43* below).

cids_using_pulsar / consumer-stat-capture.py

```
89        # truncate ip_sniffer
90        mycursor = mydb.cursor()
91        sql = "truncate ip_sniffer"
92        mycursor.execute(sql)
93        mydb.commit()
94        print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : Truncate complete for IP Sniffer")
95        mycursor.close()
96
97        for attacker_ip in block_list:
98            attacker_ip = str(attacker_ip).translate(None, "u,'()")
99            pulsar_publish(attacker_ip)
```

Figure 43 - Publish to Pulsar

Evaluation

For the purpose of this test, a topology has been created on each network. In this work, two networks have been used for evaluation of the testbed. The below table lists the various hosts and switches along with their corresponding IP addresses.

Below are the respective topologies of the two networks.

Network 1

#	Switch	Host	Host IP
1	s11	h11	10.0.0.1
2	s11	h12	10.0.0.2
3	s11	h13	10.0.0.3
4	s11	h14	10.0.0.4
5	s11	h15	10.0.0.5
6	s12	h16	10.0.0.6
7	s12	h17	10.0.0.7
8	s12	h18	10.0.0.8
9	s12	h19	10.0.0.9
10	s12	h125	10.0.0.25

Network 2

#	Switch	Host	Host IP
1	s21	h21	10.0.1.1
2	s21	h22	10.0.1.2
3	s22	h23	10.0.1.3
4	s22	h24	10.0.1.4
5	s22	h25	10.0.1.5
6	s23	h26	10.0.1.6
7	s23	h27	10.0.1.7
8	s23	h28	10.0.1.8
9	s24	h29	10.0.1.9
10	s24	h210	10.0.1.10
11	s24	h211	10.0.1.11

Table 2 - Topology Information of Two networks

Figures 44 & 45 show the graphical representation of the two networks being used for this research.

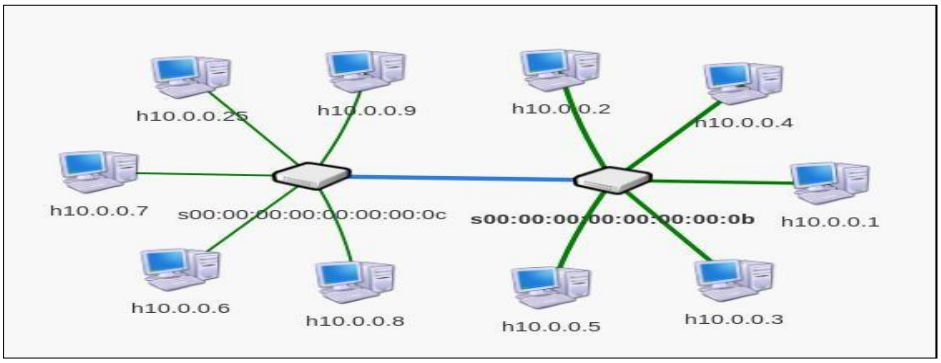

Figure 44 - Network-1 Graphical representation

ISSN No:-2456-2165

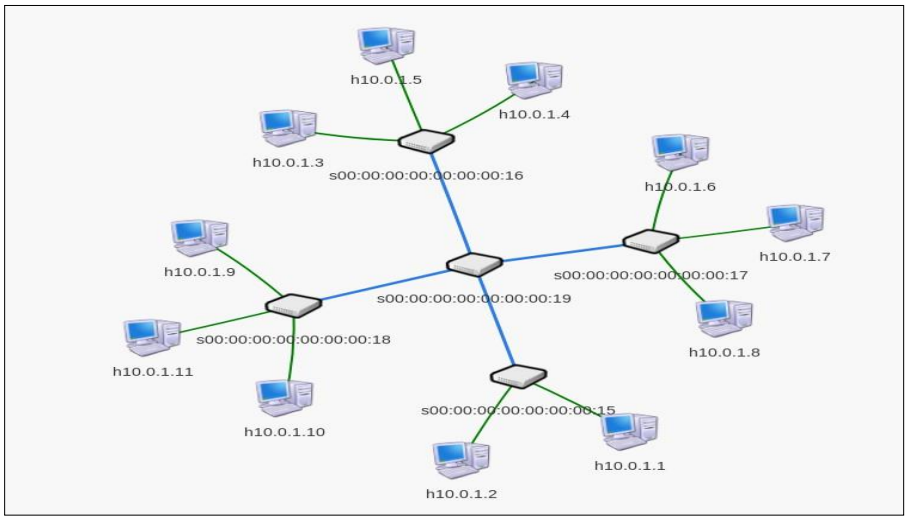

Figure 45 - Network-2 Graphical Representation

In each network, the *sniff-traffic* scripts start with the producer while the *collector-load* scripts subscribe to the same topic and load it to the respective database tables.

Below screenshot *(Figure-46)* is the output of the *collector-load* scripts, which loads the sniffed traffic from the pulsar topic to the database. It loads 5 messages instantly which can be seen below.

```
2018-09-02 15:55:49,612 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:49,622 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:49,691 INFO : Received message '10.0.0.1|10.0.0.9'
2018-09-02 15:55:49,704 INFO : Received message '10.0.0.4|10.0.0.9'
2018-09-02 15:55:49,704 INFO : Received message '10.0.0.4|10.0.0.9'
(5L, 'record inserted.')
2018-09-02 15:55:49,709 INFO : Received message '10.0.0.1|10.0.0.9'
2018-09-02 15:55:49,882 INFO : Received message '10.0.0.5|10.0.0.9'
2018-09-02 15:55:49,892 INFO : Received message '10.0.0.5|10.0.0.9'
2018-09-02 15:55:50,588 INFO : Received message '10.0.0.2|10.0.0.9'
2018-09-02 15:55:50,600 INFO : Received message '10.0.0.2|10.0.0.9'
(5L, 'record inserted.')
2018-09-02 15:55:50,617 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:50,628 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:50,715 INFO : Received message '10.0.0.4|10.0.0.9'
2018-09-02 15:55:50,724 INFO : Received message '10.0.0.4|10.0.0.9'
2018-09-02 15:55:50,725 INFO : Received message '10.0.0.1|10.0.0.9'
(5L, 'record inserted.')
2018-09-02 15:55:50,733 INFO : Received message '10.0.0.1|10.0.0.9'
2018-09-02 15:55:50,911 INFO : Received message '10.0.0.5|10.0.0.9'
2018-09-02 15:55:50,922 INFO : Received message '10.0.0.5|10.0.0.9'
2018-09-02 15:55:51,616 INFO : Received message '10.0.0.2|10.0.0.9'
2018-09-02 15:55:51,627 INFO : Received message '10.0.0.2|10.0.0.9'
(5L, 'record inserted.')
2018-09-02 15:55:51,642 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:51,657 INFO : Received message '10.0.0.3|10.0.0.9'
2018-09-02 15:55:51,740 INFO : Received message '10.0.0.4|10.0.0.9'
2018-09-02 15:55:51,754 INFO : Received message '10.0.0.1|10.0.0.9'
2018-09-02 15:55:51,755 INFO : Received message '10.0.0.4|10.0.0.9'
(5L, 'record inserted.')
2018-09-02 15:55:51,759 INFO : Received message '10.0.0.1|10.0.0.9'
2018-09-02 15:55:51,930 INFO : Received message '10.0.0.5|10.0.0.9'
2018-09-02 15:55:51,942 INFO : Received message '10.0.0.5|10.0.0.9'
```

Figure 46 - Collector Load Output

The *consumer-stat-capture* scripts check if the threshold for the number of packets per polling interval for a source or destination is breached for any of the hosts.

Destination Threshold Checks

If the destination threshold (set as 2000 as per *Table-1*) is breached, the script finds the top 3 contributors and adds them to the ACL rules.

The below screenshot *(Figure-47)* shows the count goes beyond 2000 in a 5 sec window (polling interval as per *Table-1*).

```
mysql> select * from dest_sniffer;
+----------+-----------------------------+---------------------+
| dest_ip  | count_per_polling_interval | loadtime            |
+----------+-----------------------------+---------------------+
| 10.0.0.9 |                        1985 | 2018-09-02 16:29:50 |
+----------+-----------------------------+---------------------+
1 row in set (0.00 sec)

mysql> select * from dest_sniffer;
+----------+-----------------------------+---------------------+
| dest_ip  | count_per_polling_interval | loadtime            |
+----------+-----------------------------+---------------------+
| 10.0.0.9 |                        2110 | 2018-09-02 16:29:55 |
+----------+-----------------------------+---------------------+
1 row in set (0.00 sec)
```

Figure 47 - Stats Captured - Destination Sniffer

The script then looks for the top 3 contributors and adds the IP addresses to the *ATTACKERS_TOPIC*. The *ATTACKERS_TOPIC* would be subscribed to, by all neighbouring networks and therefore, would be blocked on all the networks. The below screenshots *(Figure-48 & 49)* show the blocks added on both networks.

```
2018-09-02 16:29:55,703 INFO : Received message ''10.0.0.4''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,003 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.4' IP inserted into ACL Retention.
2018-09-02 16:29:56,007 INFO : Received message ''10.0.0.5''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,102 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.5' IP inserted into ACL Retention.
2018-09-02 16:29:56,108 INFO : Received message ''10.0.0.3''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,211 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.3' IP inserted into ACL Retention.
```

Figure 48 - Top Contributors blocked – Network 1

```
sheetaldash@sheetaldash-VirtualBox: ~/repo...  ×  sheetaldash@sheetaldash-VirtualBox: ~/repo...  ×  sheetaldash@she
sheetaldash@sheetaldash-VirtualBox:~/repos/cids_using_pulsar$ python post-acl-rules-nw2.py
2018-09-02 16:23:53,523 INFO : Connecting to Pulsar...
2018-09-02 16:23:53.524 INFO  ConnectionPool:63 | Created connection for pulsar://localhost:6
2018-09-02 16:23:53.526 INFO  ClientConnection:285 | [127.0.0.1:33754 -> 127.0.0.1:6650] Conne
2018-09-02 16:23:53.531 INFO  HandlerBase:53 | [persistent://sample/standalone/ns1/attacker,
2018-09-02 16:23:53.532 INFO  ConnectionPool:63 | Created connection for pulsar://a8a5052b369
2018-09-02 16:23:53.533 INFO  ClientConnection:287 | [127.0.0.1:33794 -> 127.0.0.1:6650] Conne
2018-09-02 16:23:53.538 INFO  ConsumerImpl:168 | [persistent://sample/standalone/ns1/attacker
2018-09-02 16:23:53,540 INFO : Created consumer for the topic persistent://sample/standalone/
2018-09-02 16:29:55,703 INFO : Received message ''10.0.0.4''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,052 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.4' IP inserted into ACL Retention.
2018-09-02 16:29:56,057 INFO : Received message ''10.0.0.5''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,170 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.5' IP inserted into ACL Retention.
2018-09-02 16:29:56,177 INFO : Received message ''10.0.0.3''
{"status" : "Success! New rule added."}2018-09-02 16:29:56,284 INFO : Post Complete
04:29PM 02-Sep-2018 : '10.0.0.3' IP inserted into ACL Retention.
```

Figure 49 - Top Contributors blocked – Network 2

As it can be seen that from the above implementations and evaluation of the script, the attack was initiated on *Network 1* but the block was added just not on Network 1 but also on Network 2 justifying a collaborative approach of detecting and mitigating the attack.

ISSN No:-2456-2165

Source Threshold Checks

Similarly, if the threshold for the source IP (i.e. 500) is exceeded, the source address would be added to the Pulsar topic and eventually blocked by all neighbouring networks.

As evident from the below *Figure-50*, a high-speed flow of packets is initiated from 10.0.1.1 to 10.0.1.9.

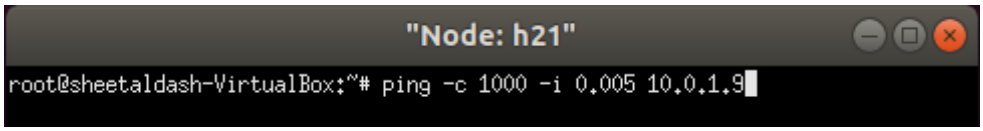

Figure 50 – High-Speed Traffic Initiated

The setting of (-i 0.005) triggers up to 200 packets per second, which exceeds the *"Threshold of requests per source"*. The corresponding source IP (in this case 10.0.1.1), is therefore blocked by all the neighbouring network when the IP address is posted to the *"ATTACKERS_TOPIC"*.

Once the source IP is blocked, the corresponding entry would be created on the *acl_retention* table on both networks, as can be seen in below screenshot *(Figure-51)*.

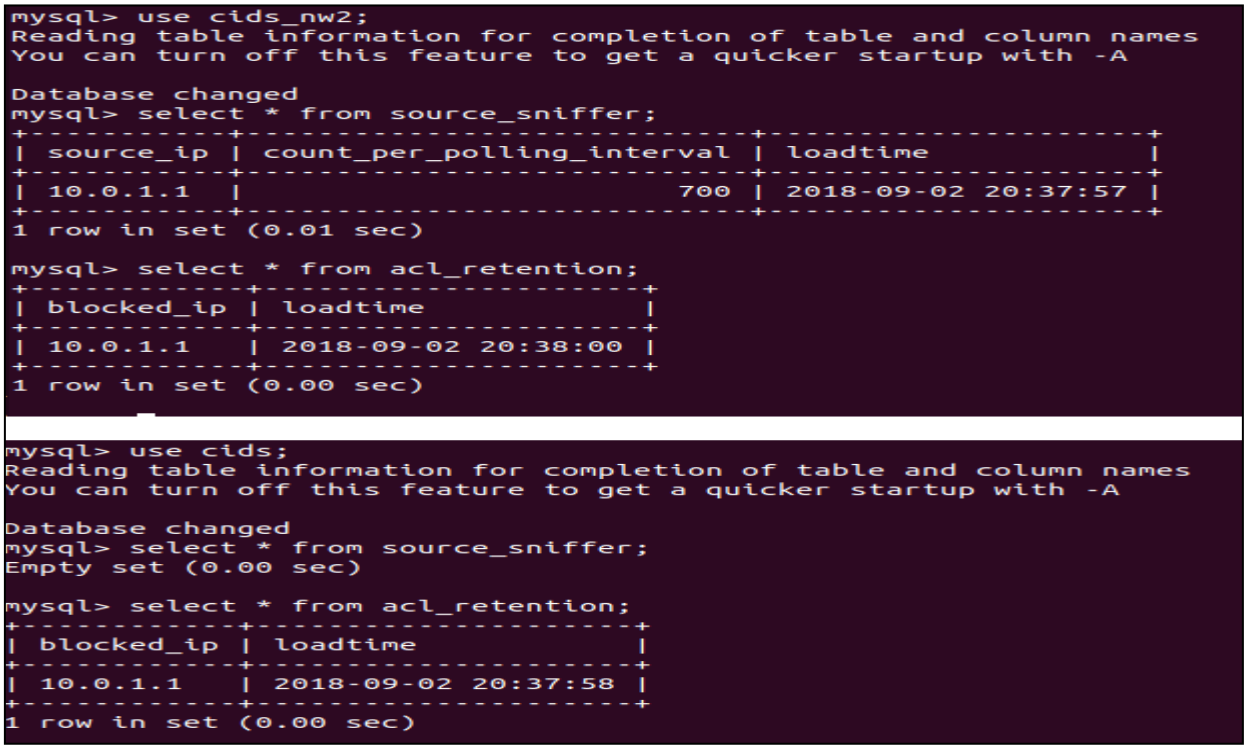

Figure 51 - ACL_RETENTION table populated for both networks

As evident from the above test, the attack was initiated on *Network 2* but the block was added on both networks due to an increase in the number of packets and exceeding a configurable limit of threshold (as it is 500 in this case).

IP Density Analysis

In this case of evaluation, the data collected on the *"ip_sniffer"* can be used to calculate the number of unique source IP addresses per destination IP that can be defined as the IP Density at a destination host.

This can be an important parameter while considering a network which is not expecting too many different IP addresses connecting to its hosts within the polling interval. In such a situation, this framework has the ability to check the IP density and block the top contributors of the servers sending a huge number of packets to the destination host.

Below screenshot *(Figure-52)* shows the query that gives the IP density for each destination host.

```
mysql> select dest_ip, count(distinct source_ip) from ip_sniffer group by dest_ip;
+----------+----------------------------+
| dest_ip  | count(distinct source_ip) |
+----------+----------------------------+
| 10.0.0.8 |                          3 |
| 10.0.0.9 |                          2 |
+----------+----------------------------+
2 rows in set (0.00 sec)
```

Figure 52 - IP Density

After having found out the destination IPs that have breached one of the thresholds that have been set, the below query would give the top contributors and accordingly those source IPs can be blocked *(Figure-53)*.

```
mysql> select source_ip,count(*) from ip_sniffer where dest_ip='10.0.0.8' group by source_ip;
+-----------+----------+
| source_ip | count(*) |
+-----------+----------+
| 10.0.0.1  |      460 |
| 10.0.0.2  |       48 |
| 10.0.0.3  |       44 |
+-----------+----------+
3 rows in set (0.00 sec)
```

Figure 53 - Top contributors to the IP with the most unique connections

Load Testing

The performance test of the framework has also been carried out. *"hping3 –faster"* or *"ping -f"* has been used to send a huge number of packets to a host and make the server overwhelmed with traffic. The below screenshot *(Figure-54)* shows ways to send high-speed packets from 10.0.0.1 to 10.0.0.8.

```
"Node: h11"
root@sheetaldash-VirtualBox:~# date;hping3 --faster 10.0.0.8
Mon  3 Sep 23:22:02 BST 2018
HPING 10.0.0.8 (h11-eth0 10.0.0.8): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=10.0.0.8 ttl=64 DF id=58339 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
len=40 ip=10.0.0.8 ttl=64 DF id=58474 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
len=40 ip=10.0.0.8 ttl=64 DF id=58475 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
len=40 ip=10.0.0.8 ttl=64 DF id=58476 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
len=40 ip=10.0.0.8 ttl=64 DF id=58477 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
```

Figure 54 - HPING3 flood attempt

As mentioned earlier, the polling interval is inversely proportional to "expected speed of traffic". Since we are expecting a very high-speed traffic, the polling interval would be reduced to 1 sec. The idea behind reducing the polling interval, even though we are more or less persisting with the same packet speed is, there can be a situation where if the polling interval is too high the DDoS attack may become successful in bringing down the target host before the speed is checked again after the waiting period. Therefore, frequent checks of the packet speed would help in stopping an attack while it is yet to cause harm to the target host as we know that timing is very important for the short intermittent DDoS attacks.

Also, in order to reduce the number of hits to the database, the frequency of database load is reduced to every 10 records instead of the previous value of every 5 records (See *Table-3*).

Threshold Name	Value
Polling Interval	1 sec
Threshold of requests per source	100 packets per second
Threshold of requests per destination	400 packets per second
LIMIT of Destination IPs block	Top 3 contributors
ACL Retention Period	15 mins
Frequency of Database Load	10 records

Table 3 - Modified thresholds for performance tests

The below screenshot *(Figure-55)* shows that in about 5-6 seconds, there was a post to the Pulsar Topic that triggered the IP to be blocked and loaded to the *acl_retention* table *(Figure-56)*. And therefore, justifying that the performance of the solution is quite high and helps in detecting the attack faster.

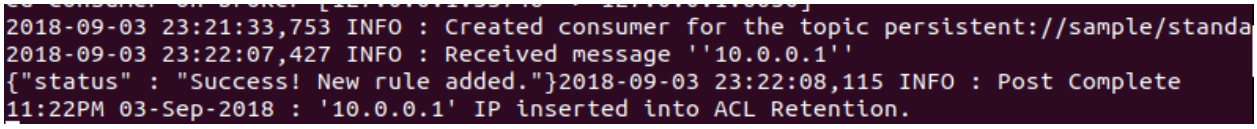

Figure 55 - DDoS Attacker identified

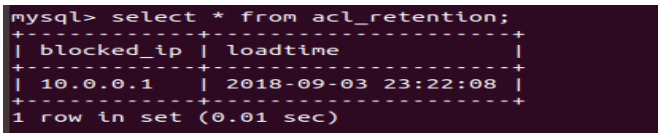

Figure 56 - Added to ACL_RETENTION

Performance Metrics

The time taken to detect an ongoing DDoS attack is defined by the Polling Interval. If the polling interval is set to 5 secs, the framework can take up to 5 secs to detect an attack. During performance benchmarking, a polling interval of one second was used and the attack was detected within a second.

Post detection, the mitigation of the attack was completed with an average turnaround time of fewer than 200 milliseconds.

#	Results	Time taken (ms)
1	2018-09-02 15:03:26,426 INFO : Received message ''10.0.0.1'' {"status" : "Success! New rule added."}2018-09-02 15:03:26,534 INFO : Post Complete	108
2	2018-09-02 15:03:26,425 INFO : Received message ''10.0.0.1'' {"status" : "Success! New rule added."}2018-09-02 15:03:26,704 INFO : Post Complete	279
3	2018-09-02 16:29:55,703 INFO : Received message ''10.0.0.4'' {"status" : "Success! New rule added."}2018-09-02 16:29:56,003 INFO : Post Complete	300
4	2018-09-02 16:29:56,177 INFO : Received message ''10.0.0.3'' {"status" : "Success! New rule added."}2018-09-02 16:29:56,284 INFO : Post Complete	107
5	2018-09-02 16:29:56,007 INFO : Received message ''10.0.0.5'' {"status" : "Success! New rule added."}2018-09-02 16:29:56,102 INFO : Post Complete	95

Table 4 - Performance Test Results

Summary

In this chapter, the author describes and implements the framework that has been proposed which includes detecting the DDoS attack, blocking them in order to mitigate the attack and followed by communicating it to other neighbouring networks for signifying a collaborative approach. The proposed solution is divided into various sections and modules like the detection mechanism, the posting of ACL rules and the housekeeping module, and it works independently. The advantage of this is that if one needs to implement any modification on a specific module the other modules remains untouched and works fine after integration. The various scripts written for this experiment has been described in detail including the reviewing of various test cases which carefully explains the launch of a DDoS attack and thereby, detecting the same, mitigating it and communicating with neighbouring networks in order to show the collaborative mechanism of this solution. Moreover, various experiments have been carried out coupled with the state of the art that has been discussed in previous chapters, which justifies the purpose of selecting specific tools and functions. These experiments include the calculation of the bandwidth usage, the specification of threshold or an upper limit, etc.

ISSN No:-2456-2165

CHAPTER 6
DISCUSSION AND FUTURE WORK

Discussion

The purpose of this study is to detect and mitigate the edge-based network attacks with a collaborative approach before they overwhelm the network of a victim. In order to define an approach and propose a framework, various existing collaborative methods of defense were studied and analysed according to their detection accuracy, performance and, scalable nature. And then, a strategic solution of a collaborative framework is proposed and a physical environment was built in order to test and analyse the distinction between the genuine and malicious packets. Various parameters such as bandwidth usage, packet speed, IP Density have been used to detect an attack. For each scenario, the functions and the mechanism of its working have been explained clearly which helps in meeting the objectives of this research (as explained in Chapter 1) and tackling with the research challenges that have been discussed before in Chapter 3.

Most traditional systems use the signature-based approach and volume limitations in order to control the network traffic. In these signature-based approaches, it is required to specify rules and signatures for detecting the known attacks whereas, the approach in this thesis helps identify attacks from unknown multiple sources. The proposed solution has shown a better performance by accurately detecting and blocking the attacks in a very short period of time, approximately 200milliseconds. The results even inferred the scalable nature of Pulsar that has been used as a pub- sub messaging system in this framework which is quite helpful for a collaborative intrusion detection system. This framework is a prototype which can be a useful solution for organizations or institutions. Hence this work can be extended further in order to build upon the research and come up with a highly robust framework.

Future Work

Below are a few areas identified where more work/research can be done, to explore possibilities.

- Pulsar provides a clustered mode which helps in enormous horizontal scalability. Pulsar installations running on multiple clustered nodes would allow for better performance as it would share the workload across multiple clusters. It would also lend more stability to the platform as it would allow for set up of secondary and failover nodes, therefore reinforcing persistent storage. This will make the solution enterprise ready.
- This framework has laid the foundation that can be extended to add more packet inspection parameters like - Packet Payload, Entropy etc.
- Machine Learning framework could possibly be used to self-maintain the thresholds with varying network load.

ISSN No:-2456-2165

CHAPTER 7
CONCLUSION

This research takes a step by step approach in providing a lightweight, easily deployable and an efficient collaborative edge-based attack detection framework. The proposed solution implements SDN integrated with the pub-sub messaging system Apache Pulsar which helps in propagating attack characteristics all the way from the victim to the attack sources in a very effective way. This is a three-step mechanism that has been implemented, the first step being the attack detection, followed by blocking it on that network server and then communicating to other networks to block the same justifying a collaborative mechanism to deal with such attacks.

This research explains the experiments that have been carried out for the prototype implementation in order to show the effect of mitigation and the instant transfer of information from one network to another about the attack. The performance benchmarking has been performed, which gave an average turnaround time to an attack of fewer than 200 milliseconds.

In addition, this solution was able to detect attacks which would come from multiple sources and would overwhelm the network with a lot of traffic. Such a detection and mitigation approach is helpful to detect unknown attack sources in the server and blocking them. The idea of polling interval that has been experimented is helpful for detecting the attacks in a very short period of time. Threshold values have been adopted based on the capacity of the system. Key consideration while setting the threshold should be – the threshold should be a limit that a genuine traffic can never touch and a malicious traffic would most definitely breach.

Like all other research mechanisms in this dynamic and complex field of study, this proposed solution can benefit from suggestions of further research and can be worked on in a larger framework to explore more on the capability of it to protect and defend against these aggressive attacks. As these attacks continue to evolve more critical analysis needs to be done and not underestimate this threat. And therefore, SDN network's integration with Pulsar can be a suitable framework to thwart emerging DDoS attacks.

REFERENCES

[1]. Ken Gray, Thomas D. Nadeau, "SDN: Software Defined Networks", [Online] Available: https://www.safaribooksonline.com/library/view/sdnsoftwaredefined/9781449342425/ch04.html?cv=1&sessionid=a6ac89a9175d47ac336b8fa73052bb54
[2]. Rainer Bye, Seyit Ahmet Campete, and Sahin Albayrak. Collaborative Intrusion Detection Framework: Characteristics, Adversarial Opportunities and Countermeasures. In Workshop on Collaborative Methods for Security and Privacy (CollSec), pages 1– 12, 2010.
[3]. Adrien Bonguet, Martine Bellaiche, "A Survey of Denial-of-Service and Distributed Denial of Service Attacks and Defenses in Cloud Computing", Future Internet MDPI, Canada, August 2017, [Online] Available: http://www.mdpi.com/1999-5903/9/3/43/pdf.
[4]. J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," SIGCOMM Computer Communication Review, vol. 34, pp. 39–53, Apr. 2004.
[5]. Bawany, N.Z., Shamsi, J.A. & Salah, K. Arab J Sci Eng (2017) 42: 425. https://doi.org/10.1007/s13369-017-2414-5.
[6]. Zargar S.T., Joshi J, Tipper D, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks", Commun. Surv. Tutor, IEEE 2013, 15, 2046–2069.
[7]. Hameed, S., Khan, H.A., "Leveraging SDN for collaborative DDoS mitigation", In Proceedings of the IEEE International Conference on Networked Systems (NetSys), Goettingen, Germany, 13–16 March 2017.
[8]. Kalkan, K., Gur, G., Alagoz, F, "Defense Mechanisms against DDoS Attacks in SDN Environment" IEEE Commun. Mag. 2017, 55, 175–179.
[9]. Y.-D. Lin, P.-C. Lin, C.-H. Yeh, Y.-C. Wang, and Y.-C. Lai, "An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention," IEEE Network, vol. 29, pp. 48– 53, May 2015.
[10]. Yu Chen, Kai Hwang, and Wei-Shinn Ku, "Collaborative Detection of DDoS Attacks over Multiple Network Domains," IEEE Trans. on Parallel and Distributed Systems, vol. 18, no. 12, Dec. 2007.
[11]. Giotis, K., Androulidakis, G., Maglaris, V., "Leveraging SDN for Efficient Anomaly Detection and Mitigation on Legacy Networks", In Proceedings of the Third European Workshop on Software Defined Networks, Budapest, Hungary, 1–3 September 2014.
[12]. Chen, X.F., Yu, S.Z., "CIPA: A collaborative intrusion prevention architecture for programmable network and SDN", Comput. Secur. 2016, 58, 1–19.
[13]. François, J., Aib, I., Boutaba, R, "FireCol: A collaborative protection network for the detection of flooding DDoS attacks", IEEE/ACM Trans. Netw. (TON) 2012, 20, 1828–1841.
[14]. Oktian, SangGon Lee, and Hoonjae Lee, "Mitigating Denial of Service (DoS) attacks in OpenFlow networks," In Information and Communication Technology Convergence (ICTC), 2014, pp. 325- 330, Oct. 2014.
[15]. Rashidi, B.; Fung, C., "CoFence: A collaborative DDOS defence using network function virtualization". In Proceedings of the 12th International Conference on Network and Service Management (CNSM), Montreal, QC, Canada, 31 October–4 November 2016; IEEE: Berlin, Germany, 2016; pp. 160–166.
[16]. S. Scott-Hayward, G. O'Callaghan and S. Sezer, "Sdn Security: A Survey", 2013 IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1-7.
[17]. "Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet." Accessed April 13, 2017. http://mininet.org/.
[18]. "What Is a Floodlight Controller? - Defined." SDxCentral, September 15, 2014. https://www.sdxcentral.com/sdn/definitions/sdncontrollers/open-source-sdn- controllers/what-isfloodlight-controller/
[19]. Yin Minn Pa Pa , Shogo Suzuki , Katsunari Yoshioka , Tsutomu Matsumoto, Takahiro Kasama, Christian Rossow, "IoTPOT: Analysing the Rise of IoT Compromises", 2015, [Online] Available: https://www.usenix.org/conference/woot15/workshopprogram/presentation/pa
[20]. Greene K, (2009), "TR10: Software-defined networking. MIT Technology Review, March/April

2009" http://www2.technologyreview.com/article/412194/tr10-software-defined- networking/
[21]. Snort homepage: https://www.snort.org/
[22]. Steven Snapp, James Brentano, Gihan Dias, Terrance Goan, Todd Heberlein, Che-Lin Ho, Karl Levitt, Biswanath Mukherjee, Stephen Smaha, Tim Grance, Daniel Teal, and Doug Mansur. DIDS (Distributed intrusion detection system) - Motivation, Architecture, and an early Prototype. In Fourteenth National Computer Security Conference, pages 167–176, 1991.
[23]. Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In IEEE Symposium on Security and Privacy (S&P). IEEE, 2002.
[24]. Frédéric Cuppens. Managing alerts in a multi-intrusion detection environment. In Annual Computer Security Applications, pages 22–31. IEEE, 2001. ISBN 0-7695-1405-7.
[25]. G. Wang, T. E. Ng, and A. Shaikh, "Programming your network at runtime for big data applications," in Proc. 1st Workshop HotSDN, 2012, pp. 103–108.
[26]. Savage, S., Wetherall, D., Karlin, A., Anderson, T.: Network support for IP traceback, IEEE/ACM Trans. Netw. VOL. 3 NO 3, pp. 226-237 (August 2002).
[27]. Yaar, A., Perrig, A., Song, D.: StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense, IEEE J. Sel. Areas Commun. VOL. 24, NO. 10, pp. 1853- 1863 (October 2006).
[28]. Law, T.K.T.,Lui,J.C.S.,Yau,D.K.Y.:YouCanRun,ButYouCan'tHide:AnEffective Statistical Methodology to Trace Back DDoS Attackers. In: Proceeding of 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2002), pp. 433-440 (2002).
[29]. Yu, Y., Zhou, W., Doss, R., Jia, W.: Traceback of DDoS Attacks Using Entropy Variations. In: Transaction on Parallel and Distributed System VOL. 22, NO. 3, pp. 412-425 (March 2011).
[30]. Floyd, S., Bellovin, S., Ioannidis, J., Kompella, K., Manajan, R., and Paxson, V.: Controlling High Bandwidth Aggregates in the Network. AT&T Center for Internet Research at ICSI (ACIRI) and AT&T Labs Research (July 2001) http://www.icir.org/pushback/pushbackJul01.pdf
[31]. Su, W., Lin T., Wu,C., Hsu J., Kuo Y. : An On-line DDoS Attack Traceback and Mitigation System Based on Network Performance Monitoring. In: Tenth International Conference on Advanced Communication Technology (ICACT), Gangwon-Do, South Korea, 17-20 Feb 2008 , pp. 1467- 1472 (2008).
[32]. Case, J.: A Simple Network Management Protocol (SNMP). IETF RFC 1157, (May 1990), http://www.ietf.org/rfc/rfc1157.txt
[33]. Chonka, A., Zhou, W., Singh, J., Xiang, Y.: Detecting and Tracing DDoS Attacks by Intelligent Decision Prototype. In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications PerCom 17-21 March 2008, Hong Kong , pp-578-583 (2008).
[34]. D. Schnackenberg, R. Balupari, D, Kindred L. Feinstein, "Statistical Approaches to DDoS Attack Detection and Response," in DARPA Information Survivability Conference and Expedition, vol. 2003, Apr.
[35]. Michie, D. and Spiegelhalter, D.J. and Taylor, C.C., Machine Learning, Neural and Statistical Classification. Ellis Horwood, 1994.
[36]. Open Networking Foundation, "OpenFlow Switch Specification - Version 1.3.5," Mar. 2015. Retrieved 17 March, 2016.
[37]. Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, Guru Parulkar, "FlowVisor: A Network Virtualization Layer", 2009
[38]. Shin, S., Gu, G.: Attacking software-defined networks: A first feasibility study. In: HotSDN, ACM (2013) 165–166.
[39]. C. Chun-Jen, et al., "NICE: Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems," Dependable and Secure Computing, IEEE Transactions on, vol. 10, pp. 198-211, 2013.
[40]. Li Gong. JXTA: A network programming environment. Internet Computing, IEEE, 5(3):88–95, 2001.
[41]. Oracle Cooperation: VitrualBox (2007), https://www.virtualbox.org/wiki/VirtualBox.

[42].Floodlight OpenFlow Controller-Project Floodlight, www.projectfloodlight.org/floodlight.
[43].Apache Pulsar, https://pulsar.incubator.apache.org
[44].Mininet Overview-Mininet, www.mininet.org/overview
[45].RahulRajewar – Github, https://github.com/rahulrajewar/cmpe210

GLOSSARY

SDN	Software Defined Networking
NIDS	Network Intrusion Detection System
CIDS	Collaborative Intrusion Detection System
IP	Internet Protocol
TCP	Transmission Control Protocol
NFV	Network Function Virtualization
DDoS	Distributed Denial of Service
ACL	Access Control List
ICMP	Internet Control Message Protocol

ISSN No:-2456-2165

APPENDIX

Code Base - https://bitbucket.org/sheet13/cids_using_pulsar/src/master

Topology Creation Script

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info from mininet.link import TCLink, Intf from subprocess import call
from os import environ """Custom topology example

Two directly connected switches plus a host for each switch: host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate our newly defined topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo
```

collector = environ.get('COLLECTOR','127.0.0.1') sampling = environ.get('SAMPLING','10')
polling = environ.get('POLLING','10')

class MyTopo(Topo):
"Simple topology example."

def_init_(self): "Create custom topo."

Initialize topology Topo._init_(self)

# Add hosts and switches
s11 = self.addSwitch('s11', cls=OVSKernelSwitch) s12 = self.addSwitch('s12', cls=OVSKernelSwitch)
h12 = self.addHost('h12', ip='10.0.0.2', defaultRoute=None, startCommand='sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h15 = self.addHost('h15', ip='10.0.0.5', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h19 = self.addHost('h19', ip='10.0.0.9', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h125 = self.addHost('h125', ip='10.0.0.25', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h17 = self.addHost('h17', ip='10.0.0.7', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h16 = self.addHost('h16', ip='10.0.0.6', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h14 = self.addHost('h14', ip='10.0.0.4', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h11 = self.addHost('h11', ip='10.0.0.1', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h13 = self.addHost('h13', ip='10.0.0.3', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
h18 = self.addHost('h18', ip='10.0.0.8', defaultRoute=None, startCommand= 'sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.default.disable_ipv6=1;sysctl -w net.ipv6.conf.lo.disable_ipv6=1')

Add links self.addLink(s11, h11) self.addLink(s11, h12) self.addLink(s11, h13) self.addLink(s11, h14) self.addLink(s11, h15) self.addLink(s12, h16) self.addLink(s12, h17) self.addLink(s12, h18) self.addLink(s12, h19)

self.addLink(s12, h125) self.addLink(s11, s12)

topos = { 'mytopo': (lambda: MyTopo()) }

nw1_normal_traffic.ksh

```
ping -c 100 10.0.0.5 &
ping -c 100 10.0.0.6 &
ping -c 100 10.0.0.7 &
ping -c 100 10.0.0.8 &
ping -c 100 10.0.0.9 &
```

nw1_attack_traffic.ksh

```
  ping -c 100 -i      0.05  10.0.0.5    &
  ping -c 100 -i      0.05  10.0.0.6    &
  ping -c 100 -i      0.05  10.0.0.7    &
  ping -c 100 -i      0.05  10.0.0.8    &
  ping -c 100 -i      0.05  10.0.0.9    &
```

check_bw_final.py

import requests import time import pulsar import logging
import sys,threading

def topology_info():
print('\n<----------SUMMARY----------
-->')
data1 =
requests.get("http://127.0.0.10:18080/wm/core/controller/summary/json") dat1 = data1.json()
number_of_switches = dat1 ["# Switches"] number_of_hosts = dat1 ["# hosts"]
print '# Switches Connected: ', number_of_switches print '# Hosts Connected: ', number_of_hosts
print '----------
\n'
return(number_of_switches , number_of_hosts)

def switch_info (number_of_switches , switch_dpids): data =
requests.get("http://127.0.0.1:18080/wm/core/controller/switches/json") dat = data.json()
k=0
for k in range(0 , number_of_switches): DPID = dat[k]["switchDPID"]
DPID = DPID.decode() switch_dpids.append(DPID)
bandwidth_one (DPID, number_of_switches)

```
return (switch_dpids)

def host_info(number_of_hosts , hosts):
a = requests.get("http://127.0.0.1:18080/wm/device/") b = a.json()
for k in range( 0, number_of_hosts):
c = str(b["devices"][k]['ipv4']) if c != []:
hosts[str(c)] = str(b["devices"][k]["attachmentPoint"])
return(hosts)

def find_host( DPID , PORT): cntrlr_summ =
requests.get("http://127.0.0.1:18080/wm/core/controller/summary/json") cntrlr_summ_json =
cntrlr_summ.json()
number_of_hosts = cntrlr_summ_json["# hosts"]
device_detail = requests.get("http://127.0.0.1:18080/wm/device/") device_detail_json = device_detail.json()
dpid_curr="1" port_curr="0" device_ipv4="1"
for index in range(0, number_of_hosts-1): try:

if(str(device_detail_json["devices"][index][u'attachmentPoint'])!='[]'):

dpid_curr=str(device_detail_json["devices"][index]['attachmentPoint'][0]['switch  '])

port_curr=str(device_detail_json["devices"][index]['attachmentPoint'][0]['port']
)
if(dpid_curr==DPID and port_curr == PORT): device_ipv4 =
str(device_detail_json["devices"][index]['ipv4']).replace('[','').replace(']',''
).replace('u','')
except:
device_ipv4="1" return device_ipv4

def pulsar_publish(attacker_ip):
ATTACKERS_TOPIC = 'persistent://sample/standalone/ns1/attacker'

# Set up for basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.INFO)
logging.info('Connecting to Pulsar...')

# Create a pulsar client instance with reference to the broker client = pulsar.Client('pulsar://localhost:6650')

# Build a producer instance on a specific topic producer = client.create_producer(ATTACKERS_TOPIC)
logging.info('Connected to Pulsar')

logging.info('Sending ACL Addition Request: %s ', attacker_ip) producer.send(attacker_ip);time.sleep(10)
```

```
client.close()


def bandwidth_one ( DPID , number_of_switches ): print DPID
for port in range(1,7):
try:
a = requests.get("http://127.0.0.1:18080/wm/statistics/bandwidth/"+DPID+"/"+str(port
)+"/json")
b = a.json() acl =
requests.get("http://127.0.0.1:18080/wm/acl/rules/json")
acl_json = str(acl.json())
bandwidth = int (b[0]["bits-per-second-rx"]) dpid=str(b[0]["dpid"])
ipv4=find_host(dpid, str(b[0]["port"])) print "\tport :", b[0]["port"]
print "\t\tBits per Second :", bandwidth

                                        if bandwidth > 20000:
                                        if ipv4.replace("\",") not in acl_json: print 'DPID: ', dpid
                                        print 'Port: ', str(port) if ipv4 != "1":
                                        print "Attacker IP : ", ipv4 pulsar_publish(ipv4)



except:
attack=False

def stat_enable():

try:
stat = requests.put("http://127.0.0.1:18080/wm/statistics/config/enable/json")
print "Enabled statistics..."

except Exception as e:
print"error while enabling statistics :" , e

def start():

try:

number_of_switches , number_of_hosts = topology_info(); switch_dpids = list()
hosts = dict() attacked_switches = set()
switch_dpids = switch_info ( number_of_switches, switch_dpids ) ; #hosts = host_info ( number_of_hosts ,
hosts );
#dpid = switch_byte ( number_of_switches, switch_dpids );

except Exception as e:
print'Error occured:', e
```

```
def main():
try:
                    stat_enable(); while 1:
                    secs = 2 time.sleep(secs) start()

except:
exit(1)

T1 = threading.Thread(target=main) T1.start()
```

sniff-traffic.py

```
from scapy.all import * import pulsar, commands
from multiprocessing import Process

SNIFFER_TOPIC = 'persistent://sample/standalone/ns1/ip_sniffer' # Set up for basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.INFO)
logging.info('Connecting to Pulsar...')
# Create a pulsar client instance with reference to the broker client = pulsar.Client('pulsar://localhost:6650')

logging.info('Connected to Pulsar') def callBack(res, msg):
print('Message published: %s'%res) def sniffPacketsWrapper(producer):
def sniffPackets(packet):          # custom custom packet sniffer action method if not (packet.haslayer(ICMP)):
if packet[IP].seq != 0: pckt_src = packet[IP].src pckt_dst = packet[IP].dst pckt_ttl = packet[IP].ttl print
"IP Packet: %s is going to %s and has ttl value %s" % (pckt_src, pckt_dst, pckt_ttl)
producer.send_async(pckt_src + "|" + pckt_dst, callBack) return
elif packet.haslayer(ICMP) and packet[ICMP].type != 0: pckt_src = packet[IP].src
pckt_dst = packet[IP].dst pckt_ttl = packet[IP].ttl print
"IP Packet: %s is going to %s and has ttl value %s" % (pckt_src, pckt_dst, pckt_ttl)
# logging.info('Source|Target: %s ', pckt_src + "|" + pckt_dst) producer.send_async(pckt_src + "|" + pckt_dst, callBack)  return
return sniffPackets


def sniffPacketsSwitch(switch):
# Build a producer instance on a specific topic producer = client.create_producer(SNIFFER_TOPIC,
max_pending_messages=100000) sniff(filter="ip",iface=switch,prn=sniffPacketsWrapper(producer))

def main():

print "custom packet sniffer"
switches=commands.getoutput('ifconfig|egrep "s1"|grep eth|awk -F":" \'{print
$1}\')
for switch in switches.splitlines(): if switch != '0':
p=Process(target=sniffPacketsSwitch, args=(switch,)) p.start()
if__name_____== '_main_': main()
```

consumer-load.py

```
import logging import sys
import os, pulsar, mysql.connector

SNIFFER_TOPIC = 'persistent://sample/standalone/ns1/ip_sniffer' SUBSCRIPTION = 'sub'
TIMEOUT = 10000
# Setup up basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.DEBUG)
counter=0 source_ips=[] dest_ips=[] traffic=[]

def insert_db(traffic): mydb=mysql.connector.connect(
host="localhost", user="root", passwd="mysql", database="cids"
)
mycursor = mydb.cursor()

sql = "INSERT INTO ip_sniffer (source_ip, dest_ip) VALUES (%s, %s)" mycursor.executemany(sql, traffic)

mydb.commit()
print(mycursor.rowcount, "record inserted.") def main(args):
global source_ips, dest_ips, traffic, counter print "Counter : "+str(counter) logging.info('Connecting to Pulsar...')

# Create a pulsar client instance with reference to the broker client = pulsar.Client('pulsar://localhost:6650')

consumer = client.subscribe(SNIFFER_TOPIC, SUBSCRIPTION, receiver_queue_size=100000)
logging.info('Created consumer for the topic %s', SNIFFER_TOPIC) while True:
try:
# try and receive messages with a timeout of 10 seconds msg = consumer.receive(timeout_millis=TIMEOUT) logging.info("Received message '%s'", msg.data()) source_ips.append(msg.data().split("|")[0])
dest_ips.append(msg.data().split("|")[1]) counter=counter+1
if counter >= 5: traffic=list(zip(source_ips,dest_ips)) source_ips=[]
dest_ips=[] insert_db(traffic) counter=0
consumer.acknowledge(msg)                    # send ack to pulsar for message consumption


except Exception: received = 0

if__name_____== '_main_': main(sys.argv)
```

consumer-stat-capture.py

```
import sys, logging import mysql.connector import time, datetime import pulsar

def pulsar_publish(attacker_ip):
ATTACKERS_TOPIC = 'persistent://sample/standalone/ns1/attacker' # Set up for basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.INFO)
```

```
logging.info('Connecting to Pulsar...')

# Create a pulsar client instance with reference to the broker client = pulsar.Client('pulsar://localhost:6650')

# Build a producer instance on a specific topic producer = client.create_producer(ATTACKERS_TOPIC) logging.info('Connected to Pulsar')

logging.info('Sending ACL Addition Request: %s ', attacker_ip) producer.send(attacker_ip)

client.close()

def insert_db(): mydb=mysql.connector.connect(
host="localhost", user="root", passwd="mysql", database="cids"
)
block_list= []

# truncate source_sniffer mycursor = mydb.cursor()
sql = "truncate source_sniffer" mycursor.execute(sql) mydb.commit()
print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y") + " : Truncate complete for Source Sniffer")
mycursor = mydb.cursor()

# truncate destination sniffer sql = "truncate dest_sniffer" mycursor.execute(sql) mydb.commit()
print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y") + " : Truncate complete for Dest Sniffer")
mycursor.close()

# Insert into Source Sniffer mycursor =  mydb.cursor() sql = "insert into source_sniffer(source_ip,count_per_polling_interval,loadtime) (select source_ip,count(1),now() from ip_sniffer where source_ip not in (select blocked_ip from acl_retention) group by source_ip)"
mycursor.execute(sql) mydb.commit()
print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : " + str(mycursor.rowcount) + " record inserted to Source_Sniffer.")
mycursor.close()

# Insert into Destination Sniffer mycursor = mydb.cursor()
sql = "insert into dest_sniffer(dest_ip,count_per_polling_interval,loadtime) (select dest_ip,count(1),now() from ip_sniffer where source_ip not in (select blocked_ip from acl_retention) group by dest_ip)"
mycursor.execute(sql) mydb.commit()
print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : " + str(mycursor.rowcount) + " record inserted to Dest_Sniffer.")
mycursor.close()

# Pick Source Ips with count > threshold mycursor = mydb.cursor()
mycursor.execute("SELECT distinct source_ip FROM source_sniffer where count_per_polling_interval>500")
source_ips = mycursor.fetchall() mycursor.close()

# Pick Destination Ips with count > threshold
```

```
mycursor = mydb.cursor()
mycursor.execute("SELECT distinct dest_ip FROM dest_sniffer where count_per_polling_interval>2000")
dest_ips = mycursor.fetchall() mycursor.close()

for ip in dest_ips:
mycursor = mydb.cursor() ip=str(ip).translate(None,"u,'()")
mycursor.execute("SELECT distinct source_ip FROM ip_sniffer where dest_ip="'+str(ip)+"' group by source_ip order by count(*) desc LIMIT 3")
ips = mycursor.fetchall() source_ips=source_ips+ips mycursor.close()
for ip in source_ips:
if ip not in block_list: block_list.append(ip)


# truncate ip_sniffer mycursor = mydb.cursor() sql = "truncate ip_sniffer" mycursor.execute(sql) mydb.commit()
print(datetime.datetime.now().strftime("%I:%M%p %d-%b-%Y")+" : Truncate complete for IP Sniffer")
mycursor.close()

for attacker_ip in block_list:
attacker_ip = str(attacker_ip).translate(None, "u,()") pulsar_publish(attacker_ip)

def main(args): while True:
insert_db() time.sleep(5)

if__name_____== '_main_': main(sys.argv)
```

post-acl-rules.py

```
import logging import sys
import os, pulsar import mysql.connector import datetime

ATTACKERS_TOPIC = 'persistent://sample/standalone/ns1/attacker' SUBSCRIPTION = 'sub1'
TIMEOUT = 10000
# Set up up basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.DEBUG)


def main(args):

mydb=mysql.connector.connect( host="localhost", user="root", passwd="mysql", database="cids"
)
logging.info('Connecting to Pulsar...')

# Create a pulsar client instance with reference to the broker client = pulsar.Client('pulsar://localhost:6650')

consumer = client.subscribe(ATTACKERS_TOPIC, SUBSCRIPTION) logging.info('Created consumer for the topic %s', ATTACKERS_TOPIC) while True:
try:
# try and receive messages with a timeout of 10 seconds msg = consumer.receive(timeout_millis=TIMEOUT) logging.info("Received message '%s'", msg.data())
```

```
mycursor = mydb.cursor()
mycursor.execute("SELECT blocked_ip FROM acl_retention where blocked_ip="+msg.data())
check_acl = mycursor.fetchall() mycursor.close()
if not check_acl:
command='curl -X POST -d \'{"src- ip":"'+msg.data()+'/32","action":"deny"}\'
localhost:18080/wm/acl/rules/json'
os.system(command) logging.info("Post Complete")

# Insert into Destination Sniffer mycursor = mydb.cursor()
sql = "insert into acl_retention(blocked_ip,loadtime) values("+msg.data()+",now())"
mycursor.execute(sql) mydb.commit()
print("{0} : {1} IP inserted into ACL Retention.".format( datetime.datetime.now().strftime("%I:%M%p %d-
%b-%Y"),
str(msg.data())))
mycursor.close()

consumer.acknowledge(msg)                    # send ack to pulsar for message
consumption
except Exception: received = 0

if__name_____== '_main_': main(sys.argv)
```

acl-rules-housekeeping.py

```
import logging import sys import os
import mysql.connector, requests, json import datetime, time

# Setup up basic logging
logging.basicConfig(format='%(asctime)s %(levelname)s : %(message)s', level=logging.ERROR)

def main(args): mydb=mysql.connector.connect(
host="localhost", user="root", passwd="mysql", database="cids", autocommit=True
)
while True: try:
mycursor = mydb.cursor()
mycursor.execute("select blocked_ip from acl_retention where loadtime < now()-interval 15 minute")
blocked_ips = mycursor.fetchall() mycursor.close()
if blocked_ips: acl_rules =
requests.get("http://127.0.0.1:18080/wm/acl/rules/json") acl_rules_data = acl_rules.content acl_rules_json =
json.loads(acl_rules_data) for ip in blocked_ips:
for i in range(0, len(acl_rules_json)): ip=".join(ip)
if acl_rules_json[i]['nw_src'].replace("/32", "") == ip: id=acl_rules_json[i]['id']
print("Attempting to remove IP - '" + ip + "' from
ACL!!")
command='curl -X DELETE -d
\'{"ruleid":"'+str(id)+'"}\' localhost:18080/wm/acl/rules/json'
os.system(command)
```

```
                                        # Clean up ACL Table mycursor = mydb.cursor()
                                        sql = "delete from acl_retention where blocked_ip =

"'+str(ip)+"'"                          mycursor.execute(sql) mydb.commit()
                                        print(datetime.datetime.now().strftime("%I:%M%p %d-
%b-%Y") + " : Deleted entry from ACL Retention table")

print("\nSleeping 5 mins!") time.sleep(300)
except Exception: received = 0

if__name_____== '_main_': main(sys.argv)
```