Addis Ababa University

College of Natural Sciences

Hybrid Threat Model of STRIDE and Attack Tree for Security Analysis of Software Defined Network Controllers

Banchiaymolu Adera Gulma

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

31, October, 2023

Addis Ababa University

College of Natural Sciences

Banchiaymolu Adera Gulma

Advisor: *Mesfin Kifle (PhD)*

This is to certify that the thesis prepared by *Banchiaymolu Adera Gulma*, titled: *Hybrid Threat Model of STRIDE and Attack Tree for Security Analysis of Software Defined Network Controllers* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name	Signature	Date

Advisor: _____

Examiner: _____

Examiner: _____

Abstract

Software Defined Network (SDN) is a network which employs software based controllers to interact with physical infrastructure and manage network traffic. It offers numerous advantages over conventional networks, including enhanced programmability, scalability and visibility. These benefits make SDN a crucial technology for addressing the evolving needs of modern networks. However, along with these benefits, SDN also introduces new security challenges due to its architectural changes. One of the main security concerns in SDN is controllers' security. Controllers serve as a core of SDN architecture, responsible for managing and controlling the network centrally. This centrality makes them high value targets for attackers and potential single points of failure. To ensure the security of SDN, it is essential to assess and mitigate vulnerabilities in SDN controllers. In previous studies, STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) threat model was used to analyze the security of SDN controllers. While it provides a systematic way of threat identification and categorization, it lacks granular and complete threat coverage and it has moderately high false negative rate. In this research, we addressed these limitations by proposing a hybrid threat model that combines STRIDE with attack tree. Attack tree provides a hierarchical and structured representation of attack scenarios and attack paths, enabling a more detailed analysis of threats. By integrating these two models, we aim to enhance the effectiveness and comprehensiveness of STRIDE only model. To evaluate our proposed hybrid model, we implemented it for security analysis of Ryu and POX controllers. As a result, we identified vulnerability to Denial of Service (DoS) attack in POX controller, which was not detected by using STRIDE only model used in previous studies. To further validate the effectiveness of our model, we conducted experimental test on mininet emulator. We exploited the detected vulnerability to launch DoS attacks on controllers and measured the impact on performance metrics bandwidth and delay. Result indicated that, both controllers are susceptible to DoS attack. However, POX controller exhibited a more significant degradation in bandwidth, a decrease of around 6.98Gbps. In contrast, the Ryu controller exhibited a decrease of around 0.74Gbps. The impact on traffic delay (jitter) was relatively small for both controllers, with values of 0.0016ms and 0.004ms for Ryu and POX, respectively. These findings show enhanced efficacy of our proposed hybrid threat model in assessing the security of SDN controllers.

Keywords: Software Defined Network, Threat Modeling, Hybrid Threat Modeling, SDN controllers, Ryu, POX

Dedication

I would like to dedicate this thesis to two special people in my life: my dad, Adera Gulma, and my little sister, Tsehay Adera. They have been my guiding lights, always supporting and believing in me. Even though they are no longer with us, their spirits continue to inspire me every day. My dad was like a mentor to me. He guided me and encouraged me to pursue my dreams. He taught me the importance of knowledge and learning. I miss him deeply, but I carry his memory in my heart as I start this academic journey. My sister, Tsehay, was always there for me, cheering me on and believing in my abilities. Losing her was incredibly difficult, and I think about her every single day. This thesis is dedicated to her, as a way to honor our special bond and the dreams we shared.

I want to express my heartfelt gratitude to both my dad and my sister for the immense impact they had on my life. Their love and belief in me have been the driving forces behind my achievements. While this dedication cannot bring them back, it is a small gesture to pay tribute to their memory and the profound influence they had on shaping who I am today. I hope that through my accomplishments, I can make them proud and carry their legacies forward. This thesis is a reflection of the love that will always connect us and a way to honor their enduring spirits.

Acknowledgments

I would like to express my deep gratitude to God for guiding and blessing me throughout my journey. I am truly thankful for the presence of the divine in my life.

I want to sincerely thank my advisor, Dr. Mesfin Kifle, for his invaluable guidance and support. His expertise and encouragement has been crucial to the success of this thesis. I am grateful for the opportunities he has provided and the knowledge he has shared.

To my wonderful mother, Eseate Hailu, thank you for your endless love, support, and inspiration. Your sacrifices and faith in me have been the foundation of my aspirations. I am forever grateful for your presence in my life.

To my beloved husband, Dawit Birhan, and my children, Maramawit and Zemichael, thank you for your love, patience, and understanding. Your support has been my strength throughout this journey. I dedicate this achievement to our shared love and the life we have built together.

Finally, I want to express my gratitude to my friends, Genet and Sr. Tenagnework Mekonin, and also to all those who have supported and believed in me. Your contributions to my personal and academic growth are deeply appreciated.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acronyms and Abbreviations

3D	3 Dimensional
AAA	Authentication, Authorization, and Accounting
AAMPA	Approach of network security Analyzing & Modeling based on Petri net & Attack
ARP	Address Resolution Protocol
CRUD	Creating, Reading, Updating, and Deleting
CVSS	Common Vulnerability Scoring System
DDoS	Distributed Denial of Service
DFD	Data Flow Diagrams
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DREAD	Damage, Reproducibility, Exploitability, Affected Users, and Discoverability.
FCSNS	Forwarding and Control Planes Separation Network Structure
FMF	Flexible Modeling Framework
GUI	Graphical User Interface
HTTP	Hypertext Transport Protocol
ICMP	Internet Control Message Protocol
IDS/IPS	Intrusion Detection Systems / a Intrusion Protection Systems
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
MAC	Media Access Control
MITM	Man-In-The-Middle
MTMT	Microsoft Threat Modeling Tool
NBI	Northbound Interface
NETCONF	Network Configuration
NFV	Network Functions Virtualization

NOS	Network Operating System
ODL	OpenDayLight
ONF	Open Network Foundation
ONOS	Open Network Operating System
OVSDB	Open vSwitch Database
OWASP	Open Worldwide Application Security Project
PASTA	Process for Attack Simulation and Threat Analysis
QTMM	Quantitative Threat Modeling Method
RADIUS	Remote Authentication Dial-In User Service
RTT	Round Trip Time
SBI	Southbound Interface
SDL	Secure Development Lifecycle
SDN	Software Defined Network
SQUARE	Security Quality Requirements Engineering
SSL	Secure Sockets Layer
STRIDE	Spoofing Tampering Repudiation Information disclosure Denial of service Elevation of privilege
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TMM	Threat Modeling Method
UDP	User Datagram Protocol
VAN	Virtual Application Networks
VAST	Visual, Agile, and Simple Threat modeling
VLAN	Virtual Local Area Network
VSC	Virtualized Services Controller
XML	Extensible Markup Language

Chapter 1: Introduction

1.1 Background

Conventional networking is a traditional method of networking that controls network traffic using fixed and specialized hardware devices like routers and switches. In this network, all three network planes such as data, control and management, are installed together in the firmware of routers and switches. Due to the hardware based architecture of this network, the device driven control plane interacting with the device driven data plane by using protocols and requires a great deal of manual administration. To configure and manage such static and hardware based network, the administrator needs to log into every network device for intervention and manage the out of box capabilities driven by hardware appliances, which need configuration changes. This administrative and technical constraint in this network makes it tedious and resource intensive [1]. In the current expanding business environment, inability to scale, network security, and performance are the main concerns.

To address these issues, software defined network (SDN) came to the field; it separates the control plane of network devices from the data plane that relays network traffic. This decoupling of network control logic from the devices that carry out the function, such as routers, which regulate the flow of information in the underlying network makes managing infrastructure simpler and provides a solution by centralizing network intelligence in the control layer and establishing an abstraction layer that logically divides the control and data planes, as shown in Figure 1. Network devices from the SDN infrastructure layer and applications from the SDN application layer are communicated to the controller via southbound and northbound interfaces respectively. SDN separates the underlying network architecture from applications by making their communication only through the controller. In this architecture, the network aims dynamically respond to shifting demands through the use of programmable packet or flow processing protocols. This can benefit networks from visibility, traffic engineering, and network virtualization and also increases network flexibility through comprehensive network management. The enabled rapid innovation in this network has the potential to significantly improve service request response times, reliability, scalability, effectiveness and flexibility that accelerates the delivery of services [2].

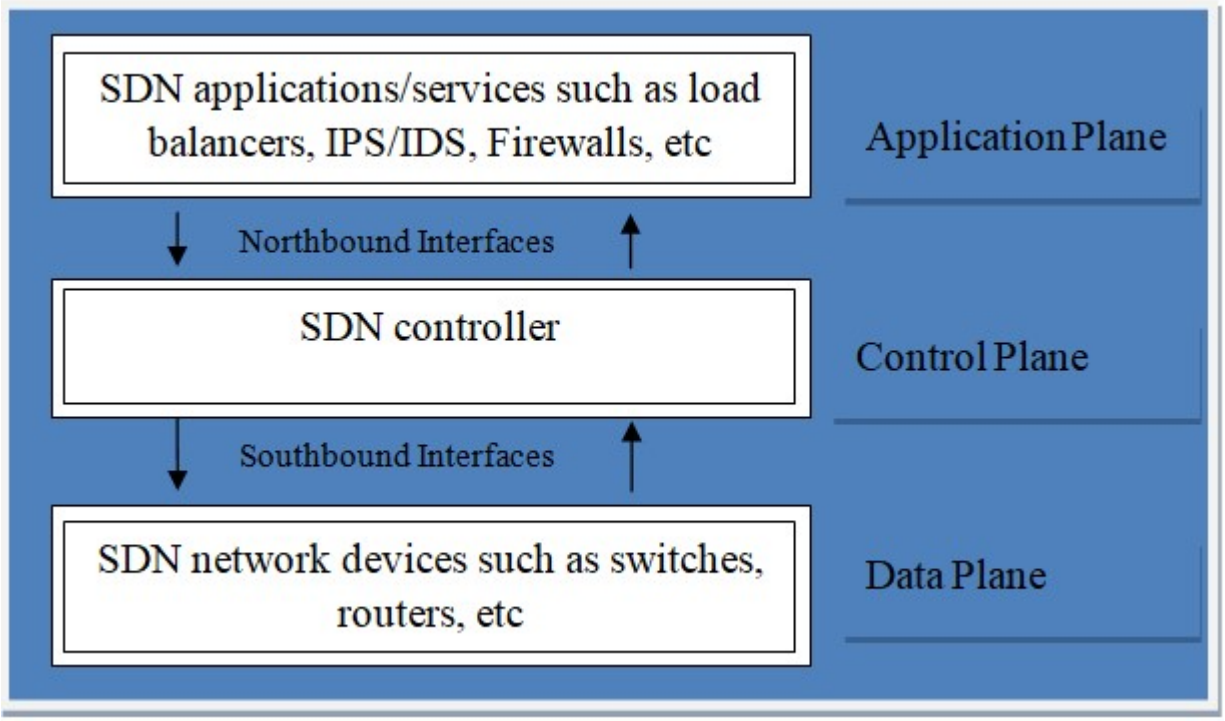


Figure 1.1 SDN architecture

✓ **Application Layer**

In this layer, typical network applications or functions such as firewall, IDS/IPS or load balancers are found without requiring specialized devices unlike conventional networks.

✓ **Control Layer**

The control layer mainly contains SDN controller which is the core of SDN networks [3] and it is the primary focus of our research. It has a global view of the whole network and its application provides flow control for better network management and application performance. SDN controllers minimize manual configurations for individual network devices by directing traffic in accordance with forwarding policies set up by a network operator.

✓ **Infrastructure Layer**

This layer is made up of physical or virtual network devices (switches, routers, etc). These devices have a role to forward the network traffic throughout their destination.

✓ **Northbound and Southbound Interfaces (NBI and SBI)**

The above three layers communicate using respective northbound and southbound interfaces. SDN applications talk to the controller through its northbound interface and the controller communicates with network devices using southbound interfaces, such as OpenFlow.

SDN controllers' centralization has a great contribution to drastic improvement of performance in the networking industry. But, it is high value target of attackers to control the entire network. Nowadays, security becomes the highest priority of current IT developments. In SDN architecture, the entire network controlled by the SDN controller becomes exposed if centralized controller is compromised. There are three ways an SDN controller might be compromised. Malicious software flaws or faults in the controller software system are to blame for the first one, the second is due to the threats coming up from rogue or corrupted controller operating programs and the final one is from threats posed by underlying network components like OpenFlow switches [4]. Centralized SDN controller is a single point of failure and in the event of an attack the attacker would have total control over the network. As a result, SDN exposes to new threats in addition to the threats already present in the conventional network due to this architectural change. One of the most frequent attacks on SDN controllers is DDoS [5]. This attack includes flooding specific network services with a large volume of fictitious traffic in order to unavailable them. The SDN controller shuts down as a result of the traffic's resource and bandwidth consumption [6] and then can take down the entire network and extremely reduce its performance. Currently, SDN controller's security takes over great attention of researchers and organizations need to implement it. The first and very important part in security is assessing vulnerability of a system. To perform security assessments there are a lot of ways, one of the most widely used way is threat modeling.

Threat Modeling

One of the key tasks in network security is threat modeling. Threat modeling is a process of creating an abstraction of a system with the intention of determining the capabilities and objectives of attackers. It is a series of planned activities for improving the security of a system by identifying threats or vulnerabilities and then defining countermeasures to prevent or mitigate the effects of threat to the system [22]. In network security the three common terms used are:

- ✓ **Threat:**

  Threat is a potential or an actual undesirable event that may be malicious like attacks on the availability of the network systems' service or incidental like the failure of critical component in a network system [60].

- ✓ **Vulnerability:**

  Vulnerability is a shortcoming that makes a threat conceivable. This can be the result of poor network system design, incorrect configuration, or inadequate and unsafe coding or network programming practices [60].

- ✓ **Attack:**

  An attack is a course of action that takes advantage of a network system's weakness or makes a threat, such as flooding a network, in an effort to obstruct service [60].

1.2 Motivation

Since the start of SDN development, the main focus of researchers has been on maintaining performance and operational flexibility of the network while separating the control plane from the data plane. In the process of accomplishing this, the security features of an SDN have been left untouched for long time. Because of this, SDN is not immune to security flaws that may occur from changes in network architecture over time or from flaws that already exist in legacy systems [20]. The network controller becomes a possible target for hackers who want to take over the entire network, despite the fact that the separation of the control plane and data plane is one step ahead towards streamlining network management. Nowadays, researchers as well as industries which are moving towards adopting SDN are greatly concerned about security issues and the resulting problems. Different researchers try to analyze SDN controllers' security but no one has used combined threat model to analyze security vulnerabilities of SDN controllers. According to the researches in ([7], [8], [21-24]) no single method can analyze the security of a system effectively. The motive of this study is to make a little contribution to this endeavor by designing the combined or hybrid threat model for analyzing the security of SDN controllers.

1.3 Statement of the Problem

In SDN architecture, the routing devices' control logic is placed in a centralized controller in order to have a single point of control. On the other hand, data plane is made up of routers, switches and other network devices that are connected to one another topologically and forward data packets in accordance with the decisions made by the central controller. With the help of this architectural modification over the conventional networks, SDN permits simple high level policies to change the network due to the removal of some device level dependencies. Moreover, from a single software console the network administrator can control the various vendor specific devices. This controller design gives it the global view of the entire network and makes it very easy to integrate additional

functionality or programs [20] and dramatically boost network performance. In contrast, the centralized SDN controller is typically the primary target for attackers because it is the central point for decisions in a network. It is also a central point of failure because it has multiple interfaces that are open to external application usage by which an intruder can exploit and gain control over the network [21]. Attackers can try to get control over the network by breaking the security of a controller. If security holes on the components of a central controller or on interfaces used to interact with other layers exist, then it can be easily compromised by the attackers. Once a controller is compromised, an attacker can gain complete control over the network. From the security aspects of SDN, controller security covers most of the security risks of this network. In order to ensure the security of SDN controllers, the first step should be assessing and identifying the security gaps (vulnerabilities/threats) in their components and communication interfaces. To perform security analysis of a system, threat modeling is the most widely used way. Previous studies on security analysis of SDN controllers by (Q. Ilyas et al., 2018) and (R. K. Arbettu et al., 2016) have used STRIDE threat model. Different researchers including Konev et al., 2022, Shevchenko et al., 2018, Scandariato et al., 2015 and Shull et al., 2023 have done researches to evaluate several threat models used in security analysis including STRIDE; all of these researches concluded that the use of single threat model cannot effectively analyze the security of a system. Studies by Scandariato et al., 2015 and Shevchenko et al., 2018 have also been conducted experimental evaluation on the effectiveness of STRIDE threat model for security assessment. As a result, they have obtained limited granularity, moderately high rate of false negative and lack of complete threat coverage as limitations of STRIDE in their study. To improve these limitations, they have recommended use of combined or hybrid threat models. To the best of our knowledge, we could not find a research which analyzes the security of SDN controllers by using hybrid threat model. According to the studies on threat modeling stated above, using this model alone is not sufficient to uncover the vulnerabilities of SDN controllers. So, in order to enhance the effectiveness of the STRIDE approach used in prior studies of our study area in [25] and [57] and to acquire better results, we propose a hybrid model fuses the best features of the two widely used threat analysis models, STRIDE and attack tree.

1.4 Objectives

General Objective

The general objective of this research is to design a hybrid threat model by combining STRIDE and attack tree threat models to perform comprehensive and granular security analysis of SDN controllers.

Specific Objectives

To achieve the general objective, the following specific objectives are identified:

- Review literatures to understand the problem, the existing threat analysis models and the main features of widely used open source SDN controllers in detail, etc.
- Design the proposed hybrid threat model.
- Implement the proposed model to identify the security threats of SDN controllers under this study.
- Create the network topology and setup a simulation environment.
- Demonstrate experimentally the identified threats or vulnerabilities by implementing test suites to show the impact of the exploitation of those vulnerabilities that will be found by our proposed model in relation to performance metrics such as bandwidth and delay of controllers.
- Evaluate and document the results.
- Compare and discuss the performance of our model with model used in previous work.

1.5 Methods

In order to achieve the objectives of this study, we will use analytical research method which includes detailed literature review in order to understand the problem, system modeling and simulation, then finally empirical evaluation. These approaches will be used to design a model to conduct a thorough, scalable, and flexible security analysis of SDN controllers.

1.5.1 Experimental Setup and Software Tools

To experiment with the proposed threat modeling method in the SDN environment, we will use different frameworks and tools [11]. Microsoft Treat Modeling Tool and Attack Tree Analysis Tool will be used to implement the treat modeling methods used in this study.

1.5.2 Emulation Environment

We will use Mininet for the purpose of network emulation and helps us to generate a virtual network with hosts, switches, and controllers and miniEdit and python programming will be used to create network topology [28]. Since Mininet runs only under the Linux operating system, the entire experiment will be carried out in Ubuntu as a virtual machine on oracle virtual box.

1.5.3 Traffic Generation Tools

We will use traffic generator tools such as iperf and hping3 [29, 30], they are open source multi feature tools to generate TCP/UDP normal traffic and attack traffic respectively.

1.6 Scope and Limitations

In this research, only centralized, open source and single controller architecture are considered. Multi-controllers, distributed controllers and commercial controller's architectures are not included in this security analysis and also security analysis of internal implementation details of the controllers is not considered. The limitations of this study will be scope of our research defined here, generalizations, assumptions and simplifications that will make throughout the this research work, the availability of data and resources (as a new technology, SDN is not implemented in most organizations in our country), integration challenges that will face during the hybridization of individual models to design our proposed hybrid model and finally time and cost constraints.

1.7 Application of Results

The application of this research result is in the research areas of SDN security by helping researchers to decide on the security mechanisms suitable to the SDN controller as well as SDN as a whole. It can be used as a baseline for selection of controllers in the development of security systems like firewall, IPS or IDS for SDN systems. It will also be used as a start point for researchers that will further study on the security area of SDN components in the future.

1.8 Organization of the Rest of the Thesis

This thesis is organized into seven chapters, including the current chapter. Chapter 2 is literature Review, which presents a comprehensive review of the existing literatures on the subject matter. It explores relevant theories, concepts, and previous studies conducted in the field. The synthesis of this literature provides a theoretical foundation for the research. Chapter 3, related work presents an analysis of works related to our research topic. It explores and discusses relevant research papers and initiatives that have been undertaken in the same or related areas. Chapter 4 presents the

proposed Hybrid Threat Model. The chapter outlines the theoretical framework, methodology, and design of the model. It explains the rationale behind the hybrid approach and how it addresses the limitations of existing models. Chapter 5 presents the implementation details of the proposed hybrid threat model to the security analysis of RYU and POX SDN Controllers. It describes the threat modeling tools used for the implementation of our proposed hybrid threat model, assumptions for performing the analysis, and the application of the hybrid model to assess the security of the controllers. The findings and results of the analysis are discussed in detail. Chapter 6 contains experimental simulation which demonstrates the exploitation of one of the main vulnerabilities results from the proposed hybrid model analysis in order to evaluate the proposed hybrid model. It contains an overview of the experimental environment used to demonstrate and test the analysis results. It also describes the tools, equipment, and configurations employed in the experiments. This chapter also presents the test results and evaluates the effectiveness of the proposed hybrid model based on the experimental findings. The implications and significance of the results are discussed. Chapter 7 will document the conclusion of the overall work. As a final chapter of this thesis study, it provides a comprehensive summary of the research findings and their implications. It discusses the contributions of the study, highlights its limitations, and suggests recommendations and avenues for future research.

Chapter 2: Literature Review

The primary goal of this chapter is to provide a detailed description on SDN, the basic difference between SDN and traditional IP network, architectural components of SDN, benefits and challenges of SDN due to its architectural change, SDN controller, basic SDN controller architectural components, SDN controllers classification, SDN controller security vulnerabilities related to its basic features, threat models used for security analysis of a system.

2.1 Software Defined Networking (SDN)

In large enterprises, the scale and complexity of networks are continually growing. Traditional IP networks require significant resources for maintenance and scalability due to their complex manageability. Extensive planning for network downtime is often necessary when performing extensive maintenance tasks. These networks consist of various network elements, including routers, switches, firewalls, network address translators, server load balancers, and intrusion detection and prevention systems. The control software used in routers and switches is typically complex, distributed, and proprietary. The network protocols employed in such networks have undergone extensive testing for interoperability and standardization over many years. Network administrators usually configure individual network devices using configuration interfaces provided by different vendors. Despite offering a centralized perspective for network configuration, some network management solutions operate at the level of individual protocols, processes, and configuration interfaces [35]. This operational approach has hindered innovation, increased complexity, and raised the capital and operational costs associated with running such networks.

However, the implementation of software-driven networking, such as SDN, has the potential to address these challenges. SDN was introduced to simplify network administration and enhance resource utilization. As defined by the Open Networking Foundation (ONF), SDN is an architecture that separates the control plane of a networking device, such as a switch or router, from its data plane. This separation enables direct programmability, agility, centralized management, adaptability, programmatically customizable capabilities, adherence to open standards, and vendor neutrality [35]. By decoupling the control plane from the data plane, SDN provides a more flexible and programmable framework for network management. It allows network administrators to have a centralized view of the network and easily implement changes. SDN promotes innovation by enabling the development and deployment of new network applications and services. Additionally,

SDN offers advantages such as increased agility, adaptability, and adherence to open standards, which can help reduce complexity and optimize costs in large enterprise networks.

2.1.1 SDN architecture

In conventional IP networks, the control and data plane are integrated as a single unit, where the control plane manages the routing table of a switch to determine the optimal path for network packets, and the data plane forwards the packets based on the control plane's instructions. In contrast, SDN architecture separates these two planes, with the control plane acting as a central controller for multiple data planes [36, 37, 38]. The SDN architecture consists of a three-layer and two-interface single controller, providing a centralized and programmable network with the following components, as indicated in Figure 2.1.

- ✓ **Infrastructure layer**

This layer comprises various networking devices that form the underlying network for forwarding network traffic. These devices perform data forwarding based on control decisions transferred from the control layer. Unlike traditional IP networks where control logic is distributed among each device, SDN has centralized control logic available at a centralized server [39]. Currently, there are numerous networking devices with OpenFlow support from the open-source community and various vendors.

- ✓ **Control layer**

The control layer consists of an SDN controller which is the core element of the SDN architecture. The controller enables centralized management, control, automation, and policy enforcement across physical and virtual network environments. It also manages a wide range of data plane resources [40]. SDN offers the potential to unify and simplify the configuration of these resources. The control layer is positioned between the application layer and infrastructure layer, and it exposes two types of interfaces: Southbound and Northbound interfaces, facilitating communication with these layers [41]. The SDN controller performs tasks such as building flow entries inside routing devices, tracking packet information or flow statistics, and more. Flow entries stored in a flow table are analogous to routing table entries in conventional IP networking. A flow table consists of three sections: matching rules, actions, and counters. Matching rules include fields from the TCP/IP header, such as MAC address, source IP, destination IP, VLAN ID, etc. Actions define operations to be performed on packets, such as forwarding packets to a specified port, dropping packets, or sending them for normal processing. The default action is to forward the packet to the controller if

no entry exists for the specific flow in the router. Once the packet is processed by the controller, it is sent back to the router along with the flow entries, and the router processes the packet accordingly. The counter field stores various counting information, such as the number of packets passing through a specific port or processed for a destination address. This information can be grouped based on a per-flow, per-table, or per-port basis [40].

✓ **Network Operating System (NOS)**

The NOS is implemented in the controller and provides fundamental features similar to an operating system. It manages input/output operations, runs programs, and provides security and safety mechanisms. Additionally, NOS offers networking features such as topology-related functions and shortest route forwarding [42]. In SDN, NOS is applied in a logically centralized manner, unlike traditional IP networks.

✓ **Application layer**

This layer consists of forwarding nodes coupled with the control decision making mechanism and includes network applications that generate network traffic. Network applications accessible at this layer are responsible for implementing the control logic, which provides appropriate commands to be installed in the data plane. The application layer enables the implementation of various networking applications, including traffic engineering, mobility and wireless, measurement and monitoring, security and dependability, and data center networking. Examples of network applications in this layer include load balancing, traffic optimization, QoS enforcement, forecasting application workloads, and fine-grained access control [43, 44].

✓ **Northbound interface**

Northbound interface is a communication interface between the control layer and the application layer in the SDN architecture. It allows network applications to interact with the SDN controller to request network services, provide instructions, and receive information about the network state. Through the Northbound Interface, network applications can utilize the capabilities of the SDN controller to implement various network functions and services. It provides a standardized and programmable interface for network applications, enabling them to access the control layer functionalities and make high-level policy and configuration decisions. This interface abstracts the underlying network infrastructure, allowing applications to define their requirements and intentions without being concerned with the complex details of network implementation. Northbound

interface can support multiple protocols or application programming interfaces (APIs) to facilitate communication between the application layer and the control layer. Some popular northbound interface protocols and APIs include REST (Representational State Transfer), and GRPC (Google Remote Procedure Call). The availability of this interface allows network applications to dynamically control and manage the network, adapt to changing traffic patterns, implement quality of service policies, optimize resource allocation, and enable network programmability. It promotes innovation and flexibility in developing network applications and services [45].

✓ **Southbound interface**

The southbound interface primarily refers to the OpenFlow protocol specification, which facilitates communication between controllers, switches, and other network nodes at lower-level components. It allows routers to identify network topology, determine network flows, and implement requests sent to it via northbound interfaces. The southbound interface installs appropriate flow rules in the switch forwarding table as determined by the controller. OpenFlow is the most widely deployed southbound standard from the open-source community. It provides information to the controller, generates event-based messages in case of port or link changes, and generates flow-based statistics for the forwarding device, which are then passed to the controller. The Packet IN message is sent when a router encounters an incoming packet for which it doesn't know how to handle. However, OpenFlow is not the only choice for the southbound interface. Other interfaces include Open vSwitch Database (OVSDB), ForCES, OpFlex etc. [46],

2.1.2 Benefits and Challenges of SDN

The decoupling of the control plane and data plane in SDN architecture brings several advantages, including improved network management, enhanced flexibility, and programmability. However, it also introduces certain challenges [47]. Let's delve into the benefits [48] and challenges [49] of SDN in more detail.

Benefits

✓ **Configuration Improvement**

SDN allows for centralized control and management of network devices from a single point, significantly improving network configuration in large-scale environments. This eliminates complexities associated with configuring heterogeneous network devices and leads to more efficient network operations.

- ✓ **Performance Optimization**

  By separating the control plane from the data plane and centralizing control, SDN simplifies performance optimization. It addresses challenges related to Quality of Service (QoS) support and end-to-end congestion that are prevalent in traditional IP networks. This enables the development and utilization of effective mechanisms to enhance system performance.

- ✓ **Cost Reduction**

  SDN shifts network management and control from network devices to a centralized software-based control plane. The network infrastructure can be managed and orchestrated by a single controller, reducing operating costs associated with traditional network management approaches.

- ✓ **Innovation Encouragement**

  SDN provides a programmable network platform, allowing for experimentation and implementation of new applications, network services, and ideas. This flexibility fosters innovation by providing a framework for developing and deploying novel network functionalities.

Challenges

- ✓ **Performance**

  While SDN offers performance improvements, it also faces certain performance challenges. For example, the flow-based approach used in SDN introduces delays due to flow setup time and the limited number of flows per second the controller can handle. Additionally, the separation of planes in SDN architecture introduces communication overhead, potentially delaying the transmission of network traffic flows.

- ✓ **Scalability**

  Scalability in SDN can be categorized into controller scalability and network scalability. Controller scalability is crucial for network scalability. With a single controller, the throughput between nodes and the controller decreases as the network size increases. In multi-controller architectures, issues related to controller-to-controller communication and controller placement arises, impacting scalability.

- ✓ **Security**

  SDN introduces security challenges due to the sharing of network traffic among multiple users and the remote control of network logic through custom software. Security risks include attacks on network devices, threats to the control plane, vulnerabilities in communication interfaces,

and the presence of bogus traffic flows. Addressing these security concerns is essential for the broader deployment of SDN.

- ✓ **Interoperability**

  To facilitate the transition from traditional IP networks to SDN, interoperability and reliability are crucial. While deploying entirely new services based on SDN technology is relatively straightforward, integrating existing network devices and appliances into an SDN environment poses challenges. Hybrid SDN architectures that incorporate traditional and SDN-enabled devices require further development to ensure interoperability and backward compatibility with existing IP routing and multiprotocol label switching (MPLS) control plane technologies.

Understanding the benefits and challenges of SDN is essential for effectively implementing and managing SDN based networks while also addressing potential limitations and risks.

2.2 SDN Controllers

In SDN architecture, the controller plays a central role in managing flow control for improved network management and application performance. The controller platform, typically running as software on a server, is separate from the network hardware and directs switches on packet transmission. By establishing forwarding policies, the SDN controller eliminates the need for manual configuration on individual network devices, simplifying and automating network management. Essentially, the SDN controller acts as the network's operating system, residing between the network devices and the applications. It facilitates communication between network devices and applications through northbound and southbound interfaces. Northbound interfaces enable communication with applications like firewalls or load balancers, while southbound interfaces, such as the OpenFlow protocol, enable communication with specific network devices. The controller configures network devices and selects the optimal network route for application traffic using southbound APIs [50].

2.2.1 SDN Controller Types

SDN controllers can be categorized based on their implementation type, availability for use, and architecture.

✓ **Types of SDN controllers based on their implementation type**

- **Centralized Controllers**

  These controllers implement all control plane logic in a single location, where a single server handles all control plane activities. Centralized controllers provide simplicity and easy management by offering a centralized point of control. However, as network traffic increases, scalability, security, performance, and availability issues may arise due to high traffic demands and limited capacity to handle data plane devices [51]. Centralized controllers work well in enterprise networks. Examples of centralized controllers include Beacon, Rosemary, Maestro, NOX-MT, Meridian, OpenDaylight, POX, and Ryu.

- **Distributed Controllers**

  These controllers distribute their control plane logic across multiple locations, with control plane activities implemented on multiple servers within the network. Distributed controllers offer advantages in scalability, security, and performance when there is an increase in demand. They address issues such as the single point of failure and scalability limitations of centralized controllers. Distributed controllers can be implemented in flat or hierarchical architectures [52].

  - **Flat Distributed Controllers Architecture**

    In a flat architecture, each controller in the network has similar responsibilities, and the network areas are divided so that each area is managed by a single controller. This architecture resolves bottlenecks because each controller only handles a small amount of traffic, ensuring availability [53].

  - **Hierarchical Distributed Controllers Architecture**

    In a hierarchical architecture, controllers have different responsibilities and are categorized as root controllers and local controllers. Local controllers, managed by the root controller, send their information to the root controller, which handles overall network management based on the information gathered from the local controllers. The root controller is selected based on capability and fair selection if multiple controllers have similar capabilities. Hierarchical architecture achieves better performance and scalability compared to flat architecture [53]. Examples of distributed controllers include Hyperflow, Smartlight, ONOS, Fleet, ONIX, and Pane.

✓ **Types of SDN controllers based on their availability for use**

- **Open Source SDN Controllers**

  These controllers have a free license, allowing everyone to use and modify them. OpenDaylight, ONOS, POX, and Ryu are examples of open-source controllers [54].

- **Commercial or Proprietary SDN Controllers**

  Proprietary or vendor-offered SDN controllers have licenses exclusively owned by the vendors. Examples of commercial SDN controllers include Cisco Application Policy Infrastructure Controller (APIC), HP Virtual Application Networks (VAN) Controller, NEC Programmable Flow PF6800 Controller, Nuage Networks Virtualized Services Controller (VSC), and VMware NSX Controller [54].

In the following section, we will describe some commonly used open-source SDN controllers:

- **OpenDaylight (ODL)**

  ODL is a highly scalable and extensible controller platform for SDN and Network Functions Virtualization (NFV). It offers a programmable, open-source approach to network automation. ODL's modular architecture allows users to adapt it to their specific requirements, choosing the features that are most important to them. It supports various southbound protocols, including OpenFlow and NETCONF, and prioritizes security with features like Authentication, Authorization, and Accounting (AAA) and automatic discovery and security of network devices and controllers [55].

- **Open Network Operating System (ONOS)**

  ONOS is a carrier-grade, Java-based open-source controller designed for high performance, scalability, and availability. It leverages distributed systems principles and supports various use cases, from data center networking to carrier networks. It also supports a range of southbound protocols, including OpenFlow and P4Runtime [56].

- **Ryu**

  Ryu is a component-based SDN controller with an open-source implementation in Python. It is widely utilized by the research community and provides clear APIs for network application development. Ryu supports multiple protocols for hardware configuration and integration. Applications can subscribe to events from the Ryu controller's event handler without requiring authentication. Its single-process, multi-threaded architecture allows new components to be

plugged in and run as new threads, making Ryu a popular choice in the research community [57].

- **POX**

The POX controller is an open-source software platform written in Python that provides a development environment for creating SDN controllers [57]. It is designed to be lightweight and flexible, allowing developers to easily build and customize their own SDN applications. POX supports the OpenFlow protocol and provides a set of libraries and modules for handling network events, managing switches, and implementing network control logic [25].

These are just a few examples of SDN controllers, and there are many other options available in the market. The choice of an SDN controller depends on factors such as specific network requirements, scalability needs, programming language preferences, and vendor support. It's important to evaluate the features, capabilities, and community support of different controllers to determine the best fit for a particular network environment.

2.2.2 Components of SDN controllers

SDN controllers are implemented in various ways, but the majority of popular controllers such as Ryu, POX, OpenDaylight, ONOS, Floodlight, and NOX follow a common architecture. This architecture consists of essential components like northbound interfaces (NBIs), core services, computational resources, eastbound/westbound interfaces, and southbound interfaces (SBIs). Figure 2.1[59] illustrates this typical architecture.

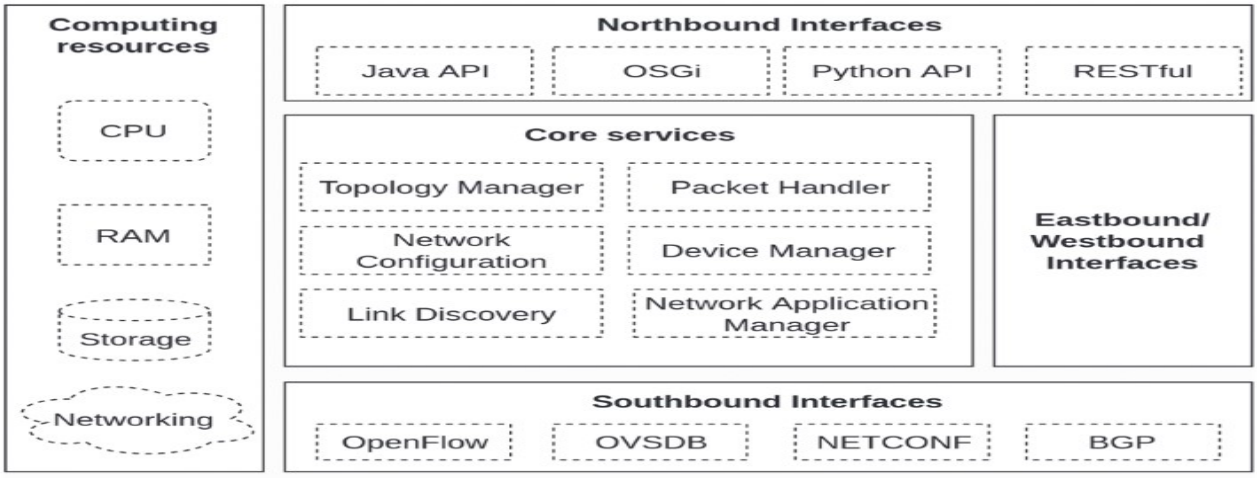


Figure 2.1 Essential components of SDN controller

The components of an SDN controller [60] can be described as follows:

✓ **NBIs and SBIs**

NBIs (Northbound Interfaces) are communication interfaces or APIs used by SDN controllers to interact with SDN applications. Similarly, SBIs (Southbound Interfaces) are communication interfaces or APIs used by SDN controllers to communicate with SDN forwarding devices.

✓ **Core services**

Core services are built-in functions within the SDN controller that provide essential services such as updating network topology information, interpreting flow rules, handling packets, configuring the network, and coordinating events among network applications.

✓ **Computing resources**

Computing resources encompass the fundamental elements required for computation, including processing power, memory, storage, and networking resources. In some SDN controller implementations like RYU, POX, or NOX, network applications are compiled and run as part of the controller module, sharing the same processing resources. However, controllers like OpenDaylight and ONOS allow runtime instantiation of network apps without restarting the controller module, but they still rely on shared resources through internal APIs.

✓ **Eastbound/westbound interfaces**

Eastbound and westbound interfaces are APIs or interfaces employed by SDN controllers for communication between distributed controllers in heterogeneous networks. Each controller typically has its own implementation of eastbound and westbound APIs.

2.2.3 SDN controller properties that make it vulnerable to attackers

As a core of SDN, SDN controller although makes networking innovation more rapid and flexible, it also exposes the network to new vulnerabilities to the attacker in addition to existing vulnerabilities in the conventional IP networks [59]. This section will cover the SDN controller's characteristics that pose new network security risks and make them vulnerable to attackers. The properties are:

✓ **Decoupling from data plane**

The separation of the control plane from the data plane in SDN architecture introduces security risks, such as man-in-the-middle attacks. While encryption solutions like TLS can mitigate these risks, the lack of their implementation in SDN allows compromised applications to potentially leak sensitive information and enables advanced attacks on flow tables, control software, and OpenFlow (OF) switch models.

✓ **Openness of SDN controllers**

Most SDN controllers are open-source, allowing developers to create network applications and middle-boxes. However, this openness also exposes SDN controllers to potential attacks if there is any vulnerability in the controller or network applications.

✓ **Programmability of SDN controllers**

SDN controllers' programmability enables third-party network applications to install new networking features using NBIs (APIs). However, this programmability can be exploited by malicious third-party apps.

✓ **Centralization of SDN controllers**

SDN controllers centralize network control, which can amplify denial of service (DoS) or distributed DoS (DDoS) attacks. The communication between the data and control planes can become a bottleneck, especially when a single controller manages multiple OpenFlow switches. Attackers can target the control plane by flooding an OpenFlow switch with various flows or by exploiting network protocols like TCP, UDP, ICMP, or IGMP.

✓ **Flow-based forwarding decisions**

SDN controllers make forwarding decisions based on flows rather than destination based routing. This flow based approach can introduce security risks such as priority bypassing attacks and conflicts in flow rules.

The above properties of SDN controllers present new network security vulnerabilities that need to be addressed. SDN controllers serve as the core and brain of SDN, but their characteristics require careful consideration and security measures to protect against potential attacks and risks.

2.2.4 SDN Controllers Selection for Security Analysis

For our security analysis, from seven open source SDN controllers (such as Beacon, Faucet, POX, ONOS, Opendaylight, Floodlight and Ryu) we have explored, two of well-known and widely used SDN controllers Ryu and POX are selected for this study based on the following features [32].

✓ **Ryu Controller**

It is an open-source, component-based SDN controller implemented in Python. The research community has made extensive use of it and it offers a set of clearly defined APIs for network application development. Because of its single process and multi-threaded architecture design, which allows for the plugging in and execution of new components as a new thread, Ryu is a controller of interest for this analysis even if it is intended for the research community[33].

✓ **POX Controller**

The POX controller is chosen for this research in part because of its simpler development environment and the reusable sample components it provides for path selection and topology discovery. It offers extensive support for current APIs. It also offers a web-based graphical user interface (GUI) written in Python, which helps to expedite the development process. It is employed in controller design, programming models, and SDN debugging. Additionally, it may identify incorrect values in ARP header fields and is also frequently utilized in research [34].

2.2.5 SDN Controllers security

SDN controllers' security is the crucial requirement in the SDN networks because they are the potential value target for the attackers to control over the entire network and also the above mentioned properties make them vulnerable to attacks, so security analysis of SDN controllers has become a vital tool to identify security threats and risks related to them to protect the entire network from failure. There are different ways used to analyze the security of SDN controllers. Threat modeling is one of the most commonly used and an effective way. Threat modeling and threat models are described in detail in the next section.

2.3 Threat Modeling

Threat modeling is a strategy for developing an abstraction of a system with the aim of identifying the skills and objectives of attackers, then utilizing that abstraction to generate and catalog potential threats that the system must be able to defend against it. Depending on their primary focus, threat modeling approaches can be divided into different categories. These approaches include those that

concentrate on the assets of the system being threat modeled, which are known as asset-centric threat modeling approaches [61], those that concentrate on the attackers, also known as attack-centric threat modeling approaches [62], and those that concentrate on the software or the system, which are known as software-centric or system-centric threat modeling approaches [62]. Considering the system being threat modeled and the available tools will help to determine which strategy to use for threat modeling.

2.3.1 Asset-centric threat modeling approaches

Asset-centric approaches focus on the assets within a system that are being threat modeled. The objective of this approach is to identify and protect an organization's critical assets. It involves evaluating the business impact or potential information loss associated with the targeted assets. Asset-centric threat modeling can also be expanded to include identifying security vulnerabilities in the system environment and understanding the goals and motivations of potential attackers [61]. This approach can help identify potential threats even if it does not specifically address bugs, insecure coding, or design practices. Examples of asset-centric threat models include VAST (Visual, Agile, and Simple Threat modeling) and Trike.

2.3.2 Attack-centric threat modeling approaches

Attack-centric threat modeling approaches focus on the perspective of the attacker. These techniques involve analyzing the methods and tools used by attackers to compromise systems. The goal is to consider system threats from the attacker's viewpoint. The primary objective of these approaches include also to identify potential attacks that could be successfully executed against a system, taking into account various known abuse scenarios and vulnerabilities [62]. An example of an attack-centric threat model is PASTA (Process for Attack Simulation and Threat Analysis) and Attack Tree.

2.3.3 System-centric threat modeling approaches

System-centric threat modeling approaches revolve around the system being analyzed. These approaches begin by considering the design of the system. The primary objective of these methods is to ensure a comprehensive understanding of the complexity of the system before examining the potential threats it may face. It is assumed that individuals involved in the system's threat modeling process possess a solid understanding of the system they are working on [62]. These approaches are commonly used for analyzing networks and systems and have become the preferred standard for

manual software threat modeling. Microsoft's Secure Development Lifecycle (SDL) framework is a notable example of a system-centric approach. This framework utilizes Data Flow Diagrams (DFDs) and STRIDE to visualize and analyze the system under consideration.

2.3.4 Hybrid threat modeling approaches

A hybrid approach combines the best features of different approaches such as asset-centric, attack-centric, and system-centric models or combines two of any combination of these strategies based on the requirement of threat modeling [62, 63]. Examples of a hybrid method are the Trike + PASTA hybrid model, STRIDE + Attack Tree hybrid model, DREAD + Attack Tree hybrid model, etc.

In the following section, some of the commonly used threat models are shortly explained:

✓ **STRIDE**

STRIDE is currently the most mature threat modeling method. It was invented by Loren Kohnfelder and Praerit Garg in 1999 and adopted by Microsoft in 2002. The name STRIDE itself is a mnemonic and stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege [9]. The description of threat types in STRIDE mnemonic along with the security properties they are violated are shown in the following table.

Table 2.1 STRIDE threat categories with their definitions and violated security properties

Threat Category	Threat Definition	Security property violated
Spoofing	Illegitimate access and use of another person's authentication information	Authentication
Tampering	Malicious modification of data	Integrity
Repudiation	Illegal or malicious operation performs in a system and then denies the involvement of the operator within the attack	Accountability
Information Disclosure	Exposure of sensitive information to persons who are not authorized to access it	Confidentiality
Denial of Service (DoS)	Attacks that prevent or deny the valid from getting service of the system	Availability

Elevation of Privileges	Unprivileged user gain privileged access then use the gained sufficient access to compromise the system	Authorization

As a system centric approach STRIDE threat modeling method requires the system model as a prerequisite. Data Flow Diagrams (DFDs) are mainly used tools to model the system. DFD is a type of diagram commonly used to describe how network architected systems work [64]. Figure 2.2 shows the four basic elements of DFDs that represent various parts of it and in addition a rectangle with the dotted lines represents trust boundaries.

Figure 2.2 Basic elements of DFD including trust boundary

The description of each parts of DFD in short is given below.

✓ Process

Process describes execution of a code in a system being modeled if any.

✓ Data store

Data store describes things such as databases, files or allocated memory segments that are used to store data in a system being modeled.

✓ Interactors

Interactors as the name suggests, is any entity that is external and interact with the system being modeled. It typically describes a user, user input or external systems or applications that either provide or retrieve information from the system.

✓ Data flow

Data flow models the flow of information in the system and describes communication between processes and data store or processes and external systems

✓ Trust boundary

Trust boundary surrounds a set of DFD elements that share the same trust level.

According to [65], each of the elements of DFD listed above has a set of threats it is susceptible to as described in Table 2.2. Table 2.2 shows which types of threats in STRIDE can be associated with

which element of DFD. This association, along with our DFD gives us a framework for investigating how our system might be attacked.

Table 2.2 Threats affecting data flow diagram elements

Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Data flows		X		X	X	
Data Stores		X		X	X	
Processes	X	X	X	X	X	X
Interactors	X		X			

STRIDE threat model has two variants such as STRIDE-per-element and STRIDE-per-interaction [66].

- STRIDE-per-element

  STRIDE-per-element is an approach where one assumes that certain threat types are more prevalent with certain elements of a DFD diagram. The STRIDE-per-element session is finished when at least one threat has been found for each type of threat element.

- STRIDE-per-interaction

  STRIDE-per-interaction approach looks at every (origin, destination, interaction) tuple and lists relevant threat types for every flow of data between elements in a diagram. The threat types that could affect the interactions are then enumerated in a table grouped by the interaction type.

✓ **PASTA**

Process for Attack Simulation and Threat Analysis (PASTA) is a risk-based threat model with seven stages (such as definition of objective, enumeration of technology, decomposition of application, threat analysis, weakness or vulnerability identification, attack modeling and then risk and impact analysis) in it and focuses on the attacker when identifying system threats. By incorporating important decision makers and requiring security input from operations, governance, design, and development, this approach elevates the threat modeling process to a

strategic level [15]. PASTA widely considered as a risk-centric framework and has an attacker-centric perspective. For analysis of threats and attack modeling, it uses attack trees.

- ✓ **Attack Tree**

Attack trees are introduced as a threat model by Bruce Schneier in 1999 [13]. The attack trees structure is a type of and-or-tree, a tree that has nodes which are connected to their children with either an AND or an OR relation. The root node of such a tree commonly represents a goal and the children represent sub-goals to the parent goal. If a parent node has a relation to its children of type AND, then all its children must have reached their sub-goal for the parent to reach its goal. On the other hand, the parent node has an OR relation to its children, then the parent will reach its goal if at least one of the child nodes reach their goal. The attack trees provide a methodical way of describing threats against, and countermeasures protecting, a system [14]. In an attack tree, the root node represents an attack or attacker goal, the children represent steps required to reach that goal and the leaf nodes represent different ways an attacker can achieve that goal. Attack trees are primarily asset-centric because they focus on identifying the assets that an attacker may target and the different ways in which they can be compromised. It is also attack centric because it focuses on the attacker. However, attack trees can also incorporate elements of data flow analysis by examining how information flows between different components of a system and where potential vulnerabilities in that flow may exist. Ultimately, attack trees are designed to provide a visual representation of the possible routes an attacker might take to compromise a particular asset, and as such they can help organizations identify and prioritize potential security risks in more granular way. Figure 2.3 shows attack tree structure.

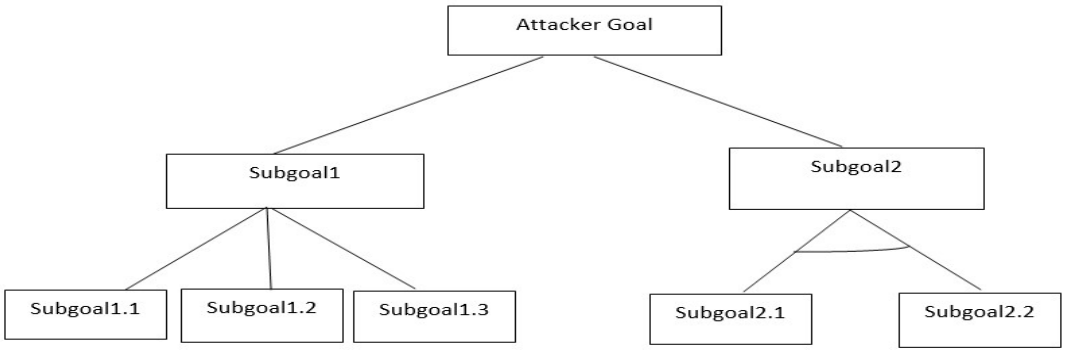


Figure 2.3 Attack Tree

- ✓ **Trike**

  Trike was created as a security audit framework in 2005 that uses threat modeling as a technique. This asset-centric threat modeling approach is utilized for risk management [62]. Like many other techniques, Trike begins by defining a system. By listing and comprehending the system's actors, assets, expected actions, and rules, the analyst must create a requirement model. This process leads to the creation of an actor-asset-action matrix, in which the rows correspond to actors and the columns to assets. Each matrix cell should be divided into four sections, one for each CRUD (creating, reading, updating, and deleting) action. The analyst should enter one of the following three values in these cells: approved action, prohibited action, or action with rules. Each cell should have a rule tree associated to it. A DFD is constructed when criteria are specified. Each element is assigned to a number of actors and assets. The analyst identifies threats as they are iterated through the DFD, which are classified as either denials of service or elevations of privilege. Trike analyzes attacks using attack trees [17] as well. An attack tree's root node is each threat that is found from the analysis. Trike employs a five-point scale for each activity, based on its probability, to evaluate the danger of attacks that could have an impact on assets through CRUD. The risks that actors are thought to pose to the asset are graded on a five-point scale (lower number = higher risk). Additionally, actors are rated on a three-dimensional scale for each action they might take on each object (always, occasionally, never). To serve as a formal technique, the Trike scale system seems too vague.

- ✓ **Hybrid Treat Modeling Method (hTMM)**

  The Hybrid Threat Modeling Method (hTMM) was developed by the Software Engineering Institute in 2018. It consists of a SQUARE (Security Quality Requirements Engineering) Method [70] to combine and develop the hybrid threat modeling from the three existing threat models Security Cards, Persona Non Grata (PnG) and STRIDE. The targeted characteristics of the model include minimal false positives, no overlooked threats, a consistent result regardless of who is doing the threat modeling, and cost-effectiveness. The model includes five steps in it such as: identifying the system to be threat-modeled, applying the Security Cards based on developer suggestions, removing unlikely PnGs, summarizing the results using tool support, continuing with a formal risk assessment method. The hTMM was applied to analyze the security of drone swarm cyber-physical system used to deliver emergency supplies for flood affected populations' [64].

- ✓ **Quantitative Threat Model (QTM)**

  QTMM is a hybrid model consists of the existing threat models Attack Trees, STRIDE, and CVSS applied in combination. It was presented by Bradley Potteiger, Goncalo Martins, and Xenofon Koutsoukos in April 2016 in Pittsburgh, Pennsylvania, at the HotSoS conference. The goal of the authors was to address the need for a threat modeling technique for cyber physical systems that featured sophisticated dependencies between their components. The initial stage in this process is to identify and construct the system's component parts. Following that, dependencies between attack category subtypes and low-level component properties are demonstrated. Scores are then determined component bases. The model also seeks to create attack ports for certain components. These attack ports, which serve as the attack trees' root nodes, show examples of actions that expose associated components to attack. Making a system risk evaluation is made easier with the score. An attack port has a high risk score and a high likelihood of being executed if it depends on a component root node with a high risk score. The inverse is also true. This method was applied in a case study for a railway communications network [67].

2.4 Hybrid Threat Model Design Approaches

According to [27] hybrid threat model is a result of the combination of two or more threat models by using variety of combining methodologies. There are a number of hybrid threat model design approaches or strategies, but the most common approaches are sequential, one model driven, integrated and iterative. In our case we propose a hybrid threat model which combines STRIDE and attack trees. In our proposed hybrid model, we selected these models based on the following criteria:

- ✓ Models maturity: Both of the selected models are the oldest and most matured [9, 14], they are well tested and have enough documentation.
- ✓ Models application area: Both of the selected models can be used in cyber only and cyber physical and physical systems [9, 14]. Both are applied in network security analysis including SDN.
- ✓ Models analysis strategy: Both of the selected models follow component based analysis [21, 22]. This feature of these models is helpful in our study its focus is on SDN controller components and their interactions.

- ✓ Models security focus or perspective: The selected methods have opposite (Defender and Attacker) security perspective [21, 22]. This feature gives complete view for our analysis, by seeing security vulnerabilities with the eye of defender and attacker.

For building a hybrid threat model by combining these two models, there are different design approaches. In this research, we can consider 4 alternative design approaches based on [27]. These approaches are:

✓ **Sequential Approach**

In this approach, we can start by creating an attack tree and then map the STRIDE threats onto the attack tree. Begin by identifying the root goal of the attack tree, and then decompose it into sub-goals and attack steps. Once the attack tree is constructed, analyze each node and determine the relevant STRIDE categories associated with the attack steps. This approach has limited threat coverage and low granularity [27].

✓ **STRIDE-Driven Approach**

In this approach, we can first analyze the system for each of the STRIDE categories independently. Identify potential threats and vulnerabilities related to STRIDE threat list. Once we have a comprehensive list of threats, we can organize them into an attack tree structure, combining related threats under common nodes for further and detailed analysis of threats associated with different attack scenarios and attack paths. This approach has comprehensive threat coverage and moderate granularity [27].

✓ **Integrated Approach**

This approach involves a simultaneous consideration of both STRIDE and attack tree elements during the design process. Start by identifying the STRIDE categories and their associated threats. Then, construct an attack tree with nodes representing the different STRIDE categories. As we populate the attack tree, consider both the STRIDE threats and the attack steps required to achieve them. This approach has comprehensive threat coverage and high granularity [27].

✓ **Iterative Approach**

This approach involves an iterative refinement process. Begin by creating a high-level attack tree or a set of STRIDE threat categories. As we explore deeper into each category or node, refine the attack tree by adding more details or decomposing nodes further. Continuously cross-reference the

STRIDE categories with the attack tree to ensure comprehensive coverage. This approach has scalable threat coverage and high granularity [27].

2.5 Summary

SDN can eliminate most of the challenges of the conventional IP networks like vendor dependency, flexibility, performance, etc. In SDN network architecture, SDN controller plays a major role as a core of the network since addressing the security issues on the SDN controllers is the primary step to secure this network. Due to the architectural changes in SDN, SDN controller has properties that make it vulnerable to variety of attacks. Analyzing the security of SDN controllers helps to differentiate the security vulnerabilities and attack vectors that generate security risks on it. There are a number of ways to analyze SDN controllers' security. One of the most commonly used ways is threat modeling. Threat modeling is accomplished by the use of threat models. A number of threat models are developed with different focus area by which their application and use is also limited to that specific area of focus. Hybrid models with different perspective make them to cover a broad area of analysis in addition to benefit from the combined strength of the individual models. There are 4 alternative design approaches for designing a hybrid threat model based on the requirement of our system we need to analyze its security.

Chapter 3: Related Work

In the field of security analysis of SDN and its components, researchers have employed various methods to identify vulnerabilities, assess risks, and develop effective security measures. This review of related work focuses on only studies that used threat modeling. There are researches which use STRIDE or attack trees or hybrid model which combines two models or either of them with other models to analyze the security of SDN and its components. Based on the model they used we will categorize them as: studies utilizing the STRIDE threat model, studies utilizing attack tree threat model and studies utilizing hybrid threat model.

3.1 Studies utilizing STRIDE Threat Model

There are studies that have utilized the STRIDE threat model method to analyze the security of SDN controllers. In [25] the authors performed the security analysis of four commonly used controllers such as FloodLight, ZeroSDN, Beacon and POX by using the STRIDE threat model presented their analysis result for each of those controllers as follows: FloodLight controller does not support TLS in its SBI that makes the controller vulnerable against tampering and information disclosure. The ZeroSDN controller did not focus on security aspects in its architecture, but in cost of security it offers best performance as its maximum throughput. Beacon controller is also vulnerable to attacks such as Tampering, Information Disclosure and DoS in its NBI. POX controller is also vulnerable to Repudiation attack due to lack of log maintenance feature in its core and NBI. Finally the authors summarized their result there is no controller which is totally secured but based on their analysis as FloodLight controller is relatively secure as compared to the other controllers in their study against different types of attacks addressed in the STRIDE threat model.

In this research, the authors use STRIDE model, it is very matured, well tested and evaluated, to identify the potential threats with defender perspective, it can model a system and classify the threats into its 6 threat categories and their work lies in providing a high-level overview of potential threats on SDN controllers. However, a limitation of relying solely on the STRIDE model is the lack of granularity in capturing detailed attack scenarios and specific vulnerabilities within SDN controllers, lack of threat coverage and moderately high rate of false negative.

The security issues of ODL, ONOS, Rosemary and Ryu SDN controllers have been analyzed in [57] by using STRIDE threat modeling technique. The authors in this study, found analysis result indicating that Opendaylight (ODL) controller effectively mitigates spoofing, tampering, repudiation, denial of service and elevation of privileges threats, however when NETCONF

protocol is used as SBI, Information disclosure of configuration files which are located in the ODL controller are vulnerable to XML external entity attack in the SBI. The authors indicated in their analysis result that open network operating system (ONOS) controller is vulnerable to known security threats like topology Spoofing and DoS attack which could be achieved through data plane or SBI. This research further investigated the security of Rosemary controller and then found unlike ODL and ONOS it has its own micro-NOS architecture which prevents it from spoofing and Elevation of privileges attacks from NBI. However, it is vulnerable to host tracking and requires additional mechanisms to protect its interactors from attacks through this vulnerability like DoS attack. According to the findings of this research, spoofing is very likely in Ryu due to its lack of authentication mechanism in both NBI and SBI and in addition it is vulnerable to elevation of privileges, repudiation of identity threats from NBI applications, information disclosure from a data flow between the controller and NBI. All the strengths and limitations indicated for the first paper are strengths and limitations of this paper because without working on different types of controllers the model and the procedures used for the analysis is almost similar in both studies.

3.2 Studies utilizing Attack Tree Threat Model

Researchers have employed the attack tree model to assess the security of SDN and its components. In [68] the authors have developed an attack tree model to analyze the potential attack scenarios in an SDN openFlow model. Their study provided a detailed analysis of attack paths and vulnerabilities, enabling the identification of critical security risks. The strength of their work lies in the detailed representation of attack paths and the ability to evaluate the potential consequences of successful attacks. But, a limitation of this work is the lack of systematic identification and analysis of individual threats.

3.3 Studies utilizing Hybrid Threat Model

Recognizing the limitations of individual analysis models, researchers have explored hybrid approaches that combine two or more types of models based on their requirements and area of application. In SDN and its components security analysis area, we only find a research in [69] which presented a security analysis of standard SBI, OpenFlow protocol, of SDN by using a hybrid model of STRIDE and attack tree threat model. The proposed model was based on sequential hybrid threat model design approach for their security assessment. By using their hybrid model for their analysis, they cannot uncover vulnerabilities such as denial of service and information disclosure which are incorporated to the openFlow protocol due to the nature of SDN. They

demonstrate how these vulnerabilities are converted into feasible attacks and then they evaluate their approach by evaluating the feasibility and impact of the attacks in their experiment. Finally, based on their analysis and evaluation, the authors recommended prevention and mitigation techniques corresponding to different network deployment and operation contexts for OpenFlow protocol.

In this research, the authors have used sequential approach which has limited threat coverage and low granularity specifically when it is implemented in systems with different components like SDN controllers [27]. By integrating attack trees with the STRIDE model, the researchers study the security of openFlow protocol which has no different components like controllers. Even if this hybrid approach has the above limitations, it allows better comprehensive understanding of the security risks involved in the protocol than the researchers obtained by using STRIDE only model in their own previous study on OpenFlow protocol security.

Authors in [70] proposed an integrated approach of attack trees and extension innovation methods from the viewpoint of attack identification for security management of SDN. First, they used attack trees to identify various attacks for security management of SDN. It consists of both the basic attack tree model and the augment attack tree model. Furthermore, they considered the use of extension innovation methods to improve attack trees by means of formalization, including application of the extension analysis method to the basic attack tree model and application of the extension transformation method to the augment attack tree model. Finally, they performed case study about a denial of service (DoS) attack on the SDN controller to validate the feasibility of proposed integrated approach based on attack trees and extension innovation methods. As a result they have exhibited the integrated method is better to effectively identify attacks than the use of attack trees alone. In this research, authors tried to solve the limitation of attack trees to analyze the security of SDN controllers by integrating it with extension innovation method such as Extenics [71].

In [72], the authors proposed a novel model for analyzing successful cyber attacks on SDN-based networks. Their model uses attack tree as a base component to gain overall knowledge about different potential attack paths and to show the attacker achievements at different stages. In this research work, the authors tried to enhance attack tree by integrating it with the mathematical models they formulated to calculate probability of occurrence for elementary attack actions and probability of successful attacks on SDN-based networks. Their model considers vulnerability severities, attack scenarios and various potential actors and their motivations. They took SDN-based

network infrastructure as an example scenario to demonstrate their new approach. Based on the results obtained from their new model, they have further inferred the most likely attack scenarios.

Authors in [73] have proposed an approach for analyzing and modeling network security for SDN called approach of network security analyzing and modeling based on petri net and attack tree (AAMPA) to analyze the security of forwarding and control planes separation network structure (FCSNS) via the combination of model and state. AAMPA divides the FCSNS structure into three parts as user access, data transmission, and control command distribution, and then builds up the models by using a hybrid approach of petri net and attack tree to represent network structure and state transferring. Finally, the authors have analyzed the security of FCSNS using Petri net and Attack tree hybrid model and then identify the security attacks and the attack paths for the three parts of FCSNS with better performance.

Works in [70], [72] and [73] used attack tree driven hybrid model which lack systematic identification and categorization of threats. This leads to lack of comprehensive threat coverage. All of these research works share this limitation.

3.4 Summary

Limitation of the studies in Section 3.1 is that using STRIDE only model lacks granularity in capturing detailed attack scenarios and specific vulnerabilities. This model is designed from a defender's perspective and does not consider how attackers can exploit vulnerabilities or the specific attack paths that indicate vulnerabilities in each system component. In addition this model has limitations like incomplete threat coverage and high rate of false negative. In Section 3.2, a study which uses the attack tree model alone lacks systematic identification and analysis of individual threats which is a time consuming process that requires a systematic approach to identify vulnerabilities in each component of the system and demands a high level of knowledge for each component in the SDN environment. In section 3.3 studies implemented hybrid models in different approaches and provide more detailed analysis of specific vulnerabilities and capture both high level threats and detailed attack scenarios. They tried to overcome the limitations of individual models and provide more comprehensive assessment of the security of the system than the individual models performs. But in these models researchers use sequential hybrid model design approach in which they have used attack tree first and then other models in their hybrid model, this design approach lacks complete threat coverage and has low granularity due to lack of systematic threat identification and categorization of attack trees. From these models, we learnt a lot of lessons for our proposed hybrid model design. In all of the studies those utilize hybrid model for security

analysis, the limitations such as lack of granularity in capturing detailed attack scenarios and specific vulnerabilities exhibited by using STRIDE threat model alone and lack of systematic identification and analysis of individual threats exhibited by using attack tree method alone have been somehow improved.

Based on the lessons learned from those studies, we have proposed a hybrid model of STRIDE and attack tree using STRIDE-Driven design approach which has comprehensive threat coverage and medium granularity to improve the limitations of STRIDE only model used in security analysis of SDN controllers in the previous researches in [25] and [57].

Chapter 4: Proposed Hybrid Threat Model for Security Analysis of SDN Controllers

4.1 Introduction

The proposed hybrid model, which combines STRIDE and attack trees represents an approach to enhance the security analysis process. This chapter introduces the concept of the hybrid model and provides a structured overview of its components and organization. The primary objective of this chapter is to present a comprehensive and granular model that integrates the strengths of both the STRIDE and attack trees threat models, aiming to improve the effectiveness of security analysis with STRIDE only model. By combining these two models, we will gain a deeper understanding of potential threats, attack paths, and vulnerabilities within SDN controllers. The chapter begins by providing a foundational explanation on what design approaches we will follow to design the proposed hybrid model. Then, the presentation of proposed hybrid threat model and explanation of its components are continued.

4.2 Hybrid Threat Model

In this study, we have designed the hybrid threat model shown in Figure 4.1 for the security analysis of SDN controllers and their communication interfaces. From the hybrid threat model design approaches discussed in Section 2.3.2, we have decided to adopt the STRIDE-driven design approach for our hybrid threat model design based on its applicability and the nature of our system under study. The integrated and iterative approaches are not suitable for our analysis due to their limited applicability. The integrated approach is typically used in complex application development processes to integrate security requirements into the design stage of the system. On the other hand, the iterative approach is applicable to large and complex systems, but it requires a high level of expertise in both methods used in the hybrid approach and a detailed system analysis and knowledge of internal implementation of the system.

We prefer and select the STRIDE-driven approach over the sequential approach to design our hybrid threat model due to the limited threat coverage and low granularity of sequential approach. It cannot improve the STRIDE only model limitations in case of our system which comprises various components and communication interfaces, each of which is susceptible to different types of attacks. If we were to use the sequential approach, it would involve creating an attack tree and mapping STRIDE threats onto it. However, in our case, the challenge lies in identifying the root node of the attack tree for each component systematically and accurately. Failure to do so would

lead to incorrect analysis results. To address this challenge, we have chosen the STRIDE-driven approach, which allows us to identify and categorize vulnerabilities in each component and interface using the STRIDE model. Then, we obtain vulnerabilities in each and every component of the controller from STRIDE model and use them along with the STRIDE attack categories, as inputs to the attack tree. This approach simplifies the analysis process and ensures comprehensive coverage and medium granularity which provides a structured representation of different threat categories, attack scenarios and attack paths in each controller component.

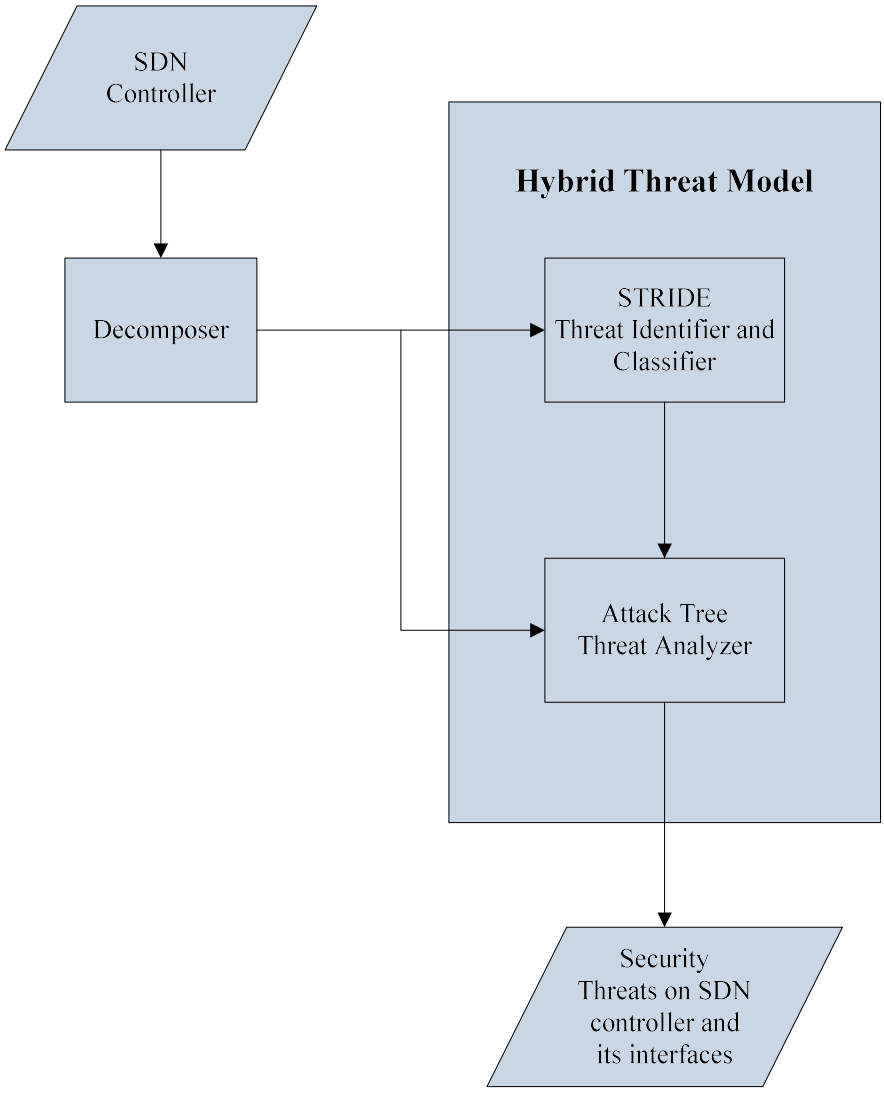


Figure 4.1 High level block diagram of the proposed hybrid threat model

4.3 Components of Hybrid Threat Model

4.3.1 SDN Controller

An SDN controller is our system; we will study its security afterwards. It is a high value asset we should have to protect from security attacks, because it is the core of SDN which manages the flow of traffic between devices by creating flow rules and passing them downwards to switches to forward traffic based on those defined rules, variety of network applications and network policies to control the overall network are implemented in it. As we have discussed its architecture in Section 2.2.2 it has different components. In this research, POX and RYU SDN controllers' are our selected study target systems based on selection criteria mentioned in Section 2.2.4 of this thesis document.

4.3.2 Decomposer

This component of our model accepts SDN controller as an input and decomposes it into different components based on data flow diagram (DFD). DFD and its 5 elements such as processes, data flows, data stores, external entities and trust boundaries are well defined in Section 2.3. Based on the concept there DFD in this component is a system level abstraction that reflects the core controller process, data flows and data stores of the SDN controller including trust boundaries as well as the external entities interacting with the SDN controller under the process of security analysis. It gives actually architectural view of the SDN controller and its communication interfaces to depict the controller system. To present the logic behind decomposer in the proposed model, we have defined the Algorithm 4.1.

Decomposer is not a part of our hybrid threat model but it is a foundation to identify the inputs for our model. In this component, our system decomposes into components such as data stores, core processes and interactors. By using algorithm 4.1, this component accepts the controller as an input and decomposes it into these components and then identify the data flows between them and trust boundaries by which different system trust levels are distinguished. In decomposer, all the data entry and exit points are identified and produces the data flow model of the controller as an output. This output is mainly used as an input for STRIDE threat identifier and classifier component of our hybrid model.

```
Input: SDN Controller
Process:
  BEGIN
    Decompose controller system into its components: controller core process, data stores and
    interactors
    Identify the entry and exit points of the system: interactions and data flows between
    components
    Identify trust boundaries
    For each component
      Define the inputs and outputs
      Map out the data flows between components
      Identify the paths that data takes through the system, including NBs and SBs
    End For
  END
Output: Controller components and data flows between them
```

Algorithm 4.1 Decomposer algorithm

4.3.3 STRIDE Threat Identifier and Classifier

This component is part of our proposed hybrid threat model used to identify and classify threats or vulnerabilities in our system under 6 STRIDE threat categories as described in Section 2.3. It uses controller DFD components from decomposer component as input and systematically iterating through all of the components to identify and categorize threats by using Algorithm 4.2 and then produces potential security vulnerabilities on each component as output that need to be further analyzed by attack trees threat analyzer component by demonstrating how an attacker exploit them to different types of security attacks.

```
Input: Controller components and data flows
Process:
  BEGIN
    Initialize an empty list to store identified threats
    For each component and data flows of the controller
      Identify the data inputs and outputs for components
      For each data flow from inputs and outputs
        Identify and classify potential threats using the STRIDE threat identifier and classifier
        If any threats are identified then
          Create a new threat object to represent the threat
          Associate the threat with the corresponding data flow
          Add threat object to the list of identified threats
        End if
      End For
    End For
  END
Output: List of identified STRIDE threats
```

Algorithm 4.2 STRIDE Threat identifier and classifier

4.3.4 Attack Tree Threat Analyzer

This component represents the different ways in which an attacker can achieve attacking of components of the SDN controller and its communication interfaces graphically. In our proposed hybrid threat model, it is used in combination with STRIDE threat identifier and classifier component. It accepts vulnerabilities on each controller component identified and classified by STRIDE and then analyzes potential attacks on such vulnerabilities of each controller component by taking ultimate goal of an attacker as a root of the attack tree to achieve each attack. In this component, Algorithm 4.3 is used to analyze security of the controller. Attack on the controller is represented by the root node of the tree, and the different ways to achieve the attack are represented by the child nodes of the root node in the analysis process. Each child node can have its own child nodes until all possible ways to achieve the goal are completely traced. This component of the proposed hybrid model can be used to discover the different ways in which an attacker could exploit those vulnerabilities of controllers in detail to prioritize security risks. Algorithm 4.3 represents in short, the steps performed in the attack tree threat analyzer.

```
Input: List of STRIDE threats, Controller components and data flows
Process:
  BEGIN
    Initialize an empty attack tree
    For each component and data flows of the controller
      Create new node in the attack tree representing the component
    End For
      For each threat in the list of STRIDE threats
        Determine the component/s of the controller that is/are affected by the threat
        For each affected component
          Find the corresponding node in the attack tree
          Create a child node under the component node to represent the threat
          Associate the threat with the component node in the attack tree
        End For
      End For
  END
Output: Security threats on SDN controller and its interfaces
```

Algorithm 4.3 Attack Tree Threat Analyzer

4.3.5 Security threats on SDN controller and its interfaces

These are security vulnerabilities pertain to various components of the SDN controller including data stores, core processes, interactors, and communication interfaces such as data flows. They are output of our hybrid threat model, which aims to comprehensively identify potential threats, vulnerabilities, and attack sequences targeting the SDN controller and its communication interfaces. They are found by leveraging Algorithm 4.4 of our proposed hybrid threat model, which initially employ the STRIDE threat identifier and classifier for identification and categorization of threats, followed by a detailed and thorough examination using the attack tree threat analyzer. These security threats are identified and isolated as potential risks and challenges associated with the overall SDN controller system.

4.4 Proposed hybrid model analysis process description

Here, we will shortly describe the steps in security analysis process of SDN controllers and their communication interfaces by using the proposed hybrid model. Algorithm 4.4 represents high level description of those steps; we will follow to implement our proposed hybrid threat model for security analysis of SDN controllers and their communication interfaces.

```
Input: SDN Controller
Process:
  BEGIN
    Decompose SDN controller
      Accept SDN controller
      Identify controller components: processes, data stores and interactors
      Define data flows and their sources and destinations, and transformations
      Analyze the interactions and dependencies between system components
    Identify and classify threats on SDN controller components and data flows
      Accept controller components and data flows
      Identify potential threats and classify them into STRIDE categories
      Map identified threats to controller components and data flows
    Analyze how identified threats exploited to attacks on controller components and data flows
      Accept controller components and data flows from decomposer and identified threats on components and
      data flows from STRIDE threat identifier and classifier
      Initialize attack tree for each controller component or data flow and identified threats pair as high level
      goals
          For each high level goal
            Decompose the goal into sub-goals
            For each sub-goals
              Decompose the sub-goal into attack steps
              Identify attack steps required to achieve the sub-goal
            End For
          End For
      Map STRIDE threats to attack steps
        For each STRIDE threat identified
        Identify the corresponding attack steps in the attack tree
        Establish a mapping between the threat and the attack step
      End For
    Identify specific mitigation measures for each attack step
  Output:
    Display the attack tree structure with high level goals, sub-goals, and attack steps
    Provide a mapping of STRIDE threats to corresponding attack steps
    Present the analysis of attack paths and security threats on SDN controller and its interfaces
END
```

Algorithm 4.4 Hybrid Treat Model

4.5 Summary

In this chapter, we designed a proposed hybrid threat model and explored its various components to assess security of SDN controllers. We focused on decomposer component which is a prerequisite for our proposed hybrid model and the two key threat models (STRIDE and attack trees) integrated in our proposed hybrid threat model.

In this model, decomposer component provides a visual representation of the system's components and data flows between components. It helps us to understand how data moves through the SDN

controller system to identify potential vulnerabilities related to data handling and communication channels. The STRIDE threat identifier and classifier component offers us a systematic approach to identify and categorize threats. It considers six threat categories: spoofing, tampering, repudiation, information disclosure, denial of service (DoS), and elevation of privilege. Attack trees threat analyzer component provides us a hierarchical representation of potential attack paths and steps required to achieve specific goals of attacks on vulnerabilities of the system. They allow for a structured analysis of attack scenarios, from high-level goals to detailed attack steps. By decomposing attacks into smaller steps, we will gain insights into potential vulnerabilities and can strategically recommend mitigation measures. In our proposed hybrid threat model, we have used STRIDE-Driven design approach and then combine the strengths of both STRIDE and attack trees. It leverages the STRIDE to identify threats and vulnerabilities specific to the SDN controller system components, and then maps those threats to corresponding attack steps in the attack tree. This proposed hybrid approach will provide a comprehensive and granular understanding of the system's security risks, including the potential impact of threats, attack paths, and recommended mitigation measures. This threat model enables the identification of vulnerabilities and mapping of threats to attack steps ultimately used to enhancing the security posture of SDN controllers.

Chapter 5: Implementation of the proposed Hybrid Threat Model

The implementation of a robust threat model is a requirement for assessing and ensuring the security of SDN controllers. This chapter contains two major sections such as presentation of threat modeling tools used to implement our proposed hybrid threat model and the implementation detail of a proposed hybrid threat model for the security analysis of RYU and POX SDN controllers. The objective of this chapter is to demonstrate the practical implementation of the proposed hybrid threat model by leveraging the strengths of STRIDE and attack tree threat models. We also aim to provide an effective approach which provides comprehensive and granular threat analysis and recommend mitigation strategies in SDN environments. In the next sections of this chapter, we will discuss the tools used to implement the proposed hybrid model and then we will present the security analysis of Ryu and POX controllers. Finally, the summary of the chapter will be presented.

5.1 Threat modeling tools used to implement the proposed hybrid threat model

As mentioned in the previous chapters of this thesis document, threat modeling involves systematically analyzing potential threats, vulnerabilities, and attack vectors to design or use appropriate security controls. To aid in this process, various threat modeling tools have been developed. However, a challenge arises from the shortage of freely available and absence of all-in-one modeling tools that support threat modeling process, which can hinder the widespread adoption and effectiveness of threat modeling practices. Threat modeling tools are software applications that assist security practitioners in conducting systematic threat assessments by using threat models. These tools have features to implement and automate threat models which help us to identify, analyze, and prioritize potential threats and vulnerabilities within a system. They enable the visualization of system components, data flows, trust boundaries, and potential attack vectors, facilitating the identification of potential weaknesses and aiding in the development of appropriate security controls. While there are several threat modeling tools available in the market, some popular examples include the Microsoft Threat Modeling Tool, OWASP Threat Dragon, and IriusRisk. These tools offer different capabilities, such as data flow modeling, threat categorization, risk analysis, and documentation of identified threats. Many existing tools either have limited functionality in the free versions or require a significant financial investment to access advanced features and capabilities. Our proposed hybrid threat model encompasses various aspects, such as data flow analysis implemented by using Algorithm 4.1, STRIDE Model it uses Algorithm 4.2 for its threat categorization and attack tree model also implement using Algorithm 4.3 to perform

detailed threat analysis. Having a single tool that integrates all these functionalities in our proposed hybrid threat model would streamline the threat modeling process, making it more efficient and user-friendly. However, such comprehensive tool is not available, and we have to use two separate tools (Microsoft Threat Modeling and AT-AT tools) to implement our proposed hybrid threat model. We will describe them in the next section.

5.1.1 Microsoft Threat Modeling Tool (MTMT)

We have used Microsoft Threat Modeling Tool 2016 for the implementation of data flow and STRIDE model parts of our proposed hybrid threat model. It is free and stable version and a powerful software tool designed to assist in the implementation of models of such type. It enables us to create data flow models by using our decomposer component that illustrate the movement of information within our system in which we present how data flows through components, processes, and external interfaces of the SDN controller by using algorithm 4.1. One of the key features of this tool is its support to incorporate the STRIDE threat model by using Algorithm 4.2, which provides a method for identifying and categorizing potential threats based on its six common attack vectors. So MTMT guides us in designing data flow model and then implementation of systematic analysis of each element of the data flow model against the STRIDE categories, helping to uncover potential weaknesses and vulnerabilities in Ryu and POX controllers in our case. Furthermore, the tool facilitates the documentation of identified threats and vulnerabilities, allowing us to assign severity levels, prioritize mitigation efforts.

5.1.2 Attack Tree Analysis Tool *(*AT-AT)

After implementing the data flow model, the data flow model of system components are given as an input to the STRIDE model, likewise after identifying and categorizing threats on both controllers given to the attack tree model. To implement attack tree model by using Algorithm 4.3, there are no free automation tools other than Attack-Defense Tree (AD) and AT-AT tools and there is no free or commercial tool to implement the hybrid model on a single all-in-one tool as we mentioned before. Based on the evaluation done on these two free tools in [76] and our system requirement we will select AT-AT tool over AD tool for our attack tree implementation. AT-AT is an Attack Tree Analysis Tool which is a specialized software tool used to construct attack trees based on the given algorithm from the analyst. With this tool, we will create attack trees that visually depict the different attack paths and potential combinations of attack vectors that attackers might employ on SDN controllers and their communication interfaces by using Algorithm 4.3. This tool offers an

intuitive interface for defining nodes representing specific attack steps. Here, we will manually give STRIDE threat categories we can find from the MTMT analysis result and then connect nodes to illustrate the relationships between different attack steps and their dependencies. The AT-AT tool also allows for the quantification and assessment of attack paths by assigning probabilities, impact values, and risk factors to each node in the attack tree [79]. This enables the evaluation of the overall risk associated with specific attack scenarios, aiding in the prioritization of security measures and resource allocation for mitigation. Finally, by utilizing the Microsoft Threat Modeling Tool for data flow modeling and threat identification and classification with STRIDE, and subsequently employing the AT-AT tool for constructing attack trees, then we will gain a comprehensive approach to threat assessment in SDN controllers. The combination of these tools enables the identification of potential threats through data flow analysis and the evaluation of the likelihood and impact of those threats through attack tree modeling. This integrated approach implements our proposed hybrid threat model which helps to analyze and recommend the security measures to strengthen the security posture of SDN controllers and to safeguard these critical components of the SDN infrastructure.

After installing prerequisite software such as visual studio code and making java react configurations in the software, the commands in Figure 5.1 are used to open the AT-AT tool on our browser.

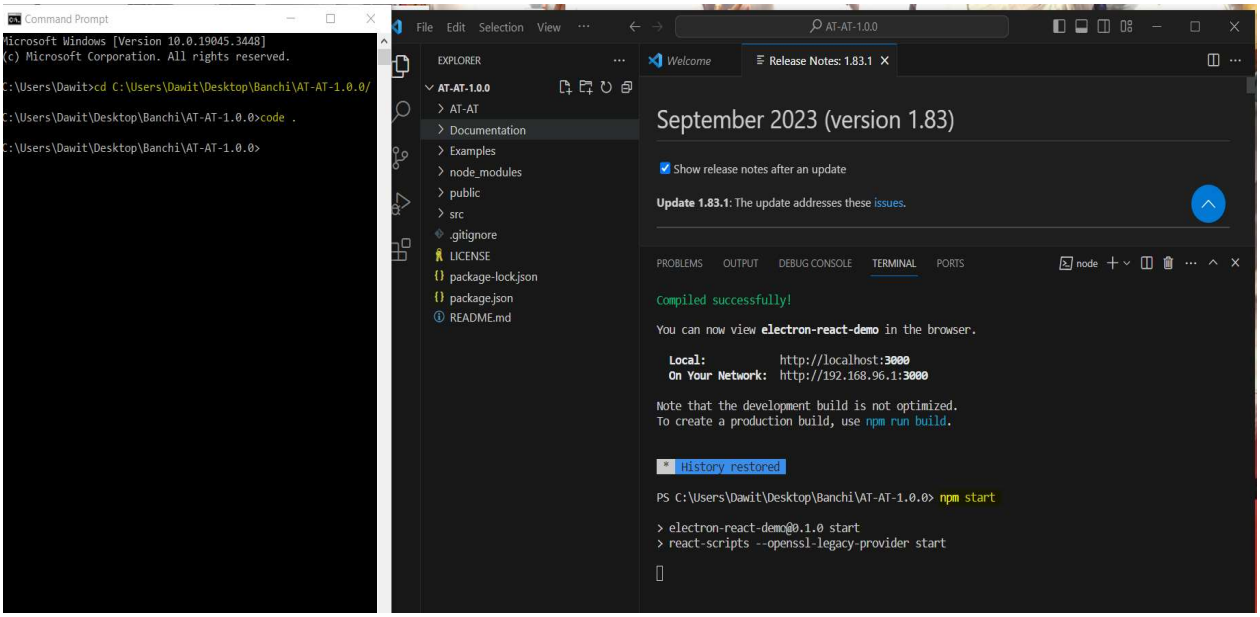


Figure 5.1 Commands used to start up the AT-AT attack tree threat modeling tool

In Figure 5.1, after running "*npm start*" command in visual studio code terminal window shown in the right side of the figure which is automatically opened by using "*code* ." command in the command prompt window with the path of the AT-AT software package folder shown at the left side of the figure, then browser based AT-AT tool will open in our browser and then generate attack tree for each of the controller component and threat category pair accepted from the STRIDE method as output and manually insert to it as input by using Algorithm 4.3.

5.2 Implementation of proposed hybrid threat model for security analysis of controllers

5.2.1 Data Flow Diagram/ Model

In this section, we will model our system data flows, processes, data stores and interactors by using decomposer component of our proposed hybrid threat model. To make clear our system, we need to create and model a simple, general case scenario, namely accessing a file, when a user or an application wants to access a file on a server that is located in an infrastructure layer of the network. There are several ways the data flow to/from a controller which could be exploited by an intruder to compromise the security of a controller. Data flows between two controllers shall face multiple security issues including inter-controller trust, and inter-domain trust when controllers are placed in a different domain. In this work, a single controller model is used for the analysis. So, we assume that data flow between controllers is not considered. The model consists of two clients, independent from the system, identical and not trusted, users/ applications from application layer which are connected to the controller through user/application interface NBIs and the controller connected to the infrastructure layer devices through openFlow protocol or SBIs.

In this scenario, when a user wants to access a file on a server located in a different part of the network, the user's computer sends a request to the application layer of the SDN controller through NBIs. The application layer of the controller determines the best path for the traffic to reach the server. Then, it sends a flow rule to the infrastructure layer of the controller. The infrastructure layer of the controller temporarily writes the flow rule in its storage and sends the flow rule to the switch that is closest to the user's computer through SBIs. The switch updates its flow table to reflect the new flow rule and then it forwards the user's request to the next switch in the path to the server. This process continues until the request reaches the switch that is closest to the server. The switch that is closest to the server forwards the request to the server. The server accepts the request and then sends the file to the user's computer. The flow rule is deleted from the controller's

infrastructure layer. The server sends a status update to the closest switch and then the switch sends it to the controller through SBIs. The controller sends the status update to the application layer of the user's computer. The application layer of the user's computer displays the status update to the user. This is a high level description of the scenario to visualize the data flow between the clients and the system we modeled.

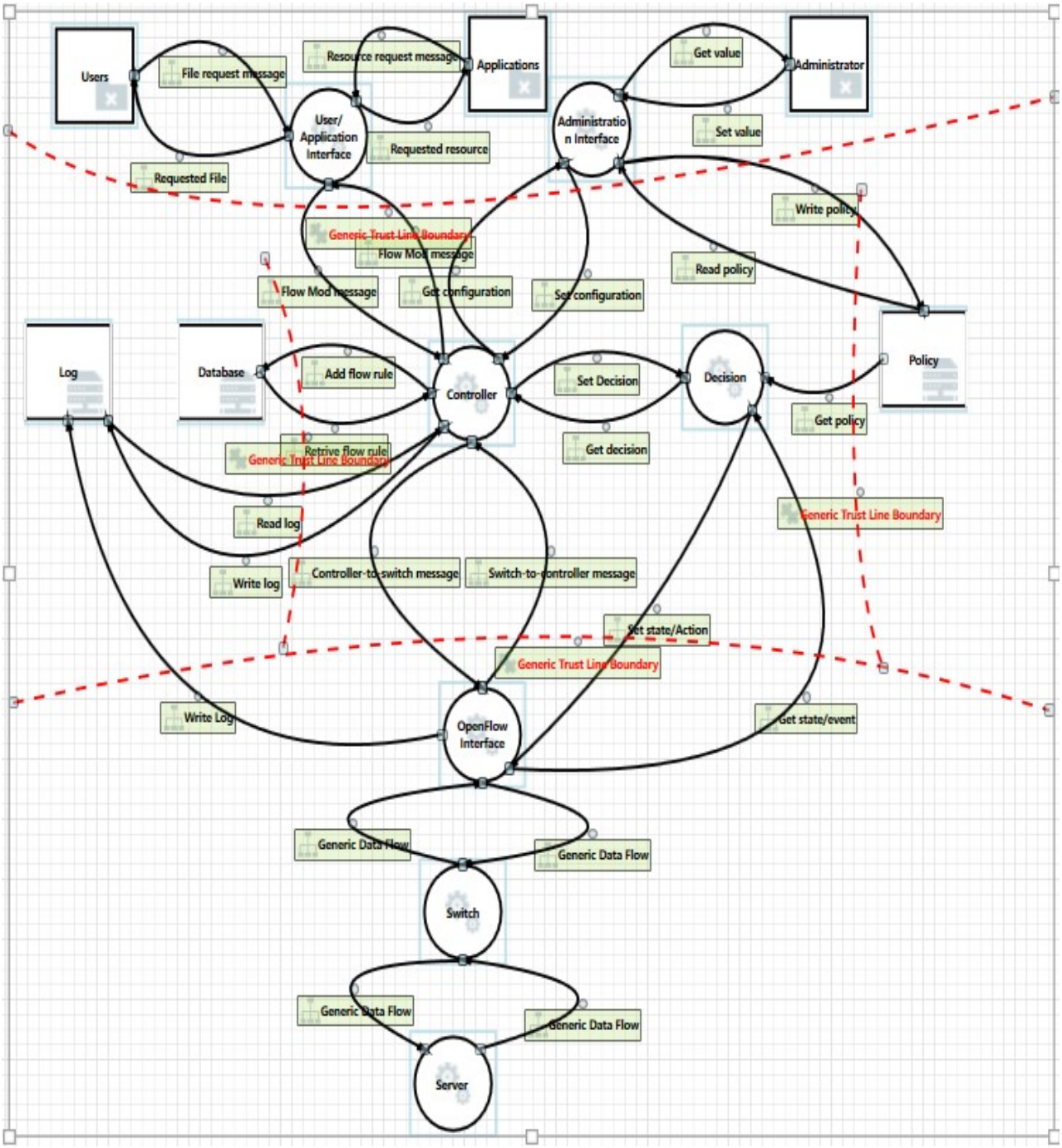


Figure 5.2 Data flow model of the controller for file request scenario

The data flow model in Figure 5.2 is a representation of a scenario we have created above to make clear the whole SDN system but it is very detailed, complicated and difficult to implement for our security analysis. Therefore, we will simplify it by considering only important components of the data flow model for our analysis by eliminating unnecessary and out of our analysis scope components and produce a simplified data flow model shown in Figure 5.3.

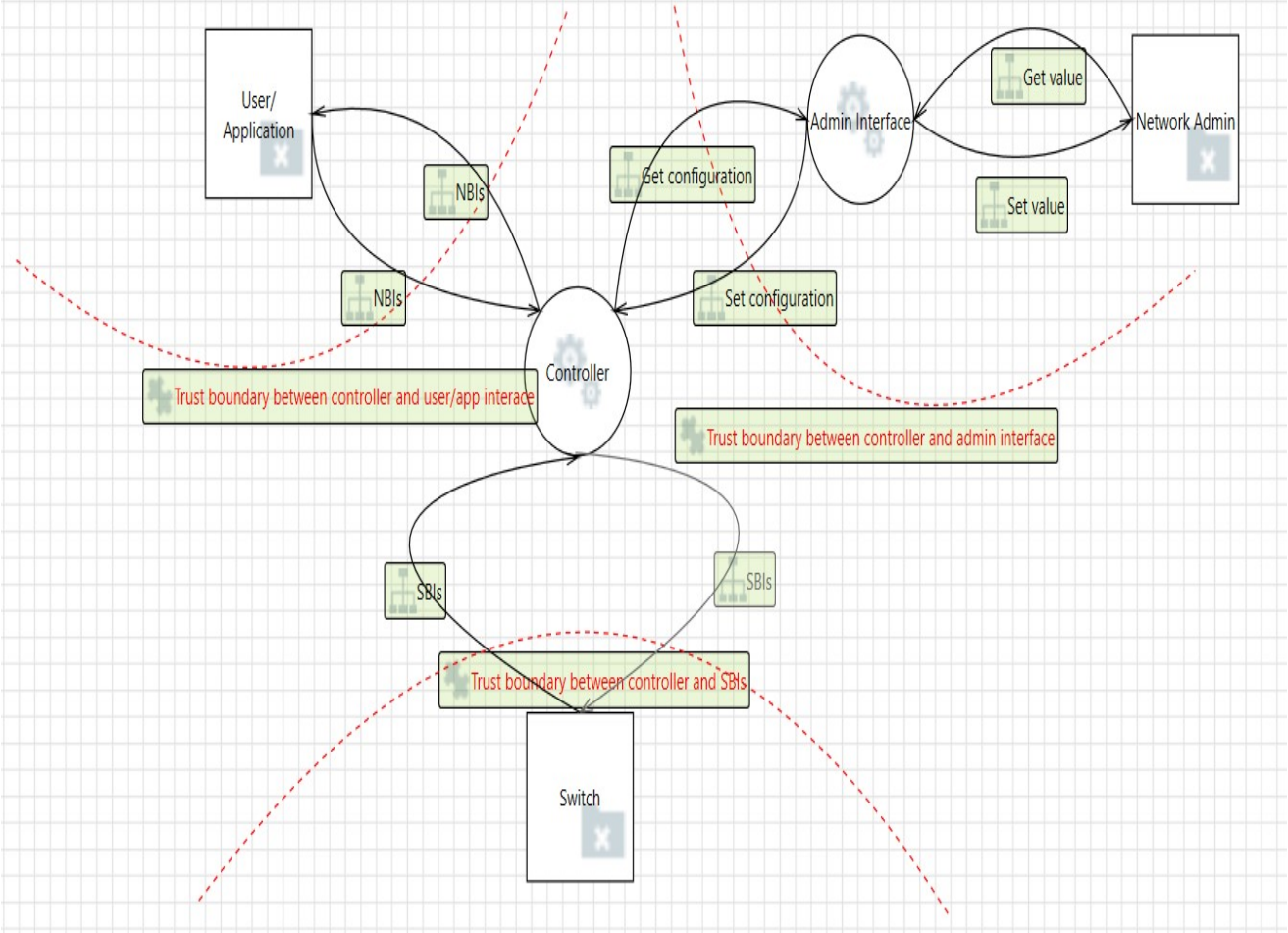


Figure 5.3 Simplified Data Flow Model

5.2.2 STRIDE Threat Model Security Analysis

In this analysis, the data flow model of a system shown in Figure 5.3 is used as an input. This data flow model simply and clearly shows several ways the data flow to/from a controller and controller components which could be exploited by an intruder to compromise the security of a controller. As mentioned earlier in our research scope, only a single controller model is used for the analysis. The security analyses of multi-controller and hierarchical controllers are not considered here. The

controllers have a collection of processes that implement and carry out crucial networking tasks including topology management, load balancing, access control, etc. To examine the security of a controller which includes security evaluations of these processes are essential. Since the three layers of an SDN are completely decoupled from one another, a third party interface is required for communication between them. These interfaces include applications and controller, network devices and controller communication interfaces which are vulnerable to well-known threat categories such tampering, information exposure, and denial of service attacks (DoS). Hence, to verify the confidentiality, integrity and availability of information exchange, data flows will also be considered in the analysis.

We shall use a single DFD as shown in Figure 5.3 for both controllers under this research because most controllers functionally follow a generalized data flow mechanism. Due to the fact that each controller, despite adhering to a common design and having a similar DFD, has attained a different level of maturity for each of the security features, two distinct STRIDE threat matrices will be developed and compared for each controller. We may generate a list of potential vulnerabilities using the STRIDE Model by taking into account the data flow model in Figure 5.3 and Algorithm 4.2. For each element, certain security issues must be considered as mentioned in Section 2.3.1 of this thesis document in which each of the DFD elements associated with list of potential vulnerabilities. Here, not all elements of the data flow model will be considered. Data flows such as internal data flows in the device or function calls in the control software and the internal data structures such as data store in the system do not cross a trust boundary, so they are not directly exploited, thus they will not be considered in the analysis. Similarly, any external system that does not directly interacting with our system is out of our research scope. Thus, they are not also included in vulnerability analysis. The controller core and the communication channels with interactors will be examined in detail. In addition, in the security analysis of individual controllers and their communication interfaces under our study, we have to use literatures and controllers documentations as a source of security information used in the STRIDE threat model throughout this thesis document.

- ✓ **Ryu controller**
  - **Data flows**

    According to Section 2.3.1 data flows are exposed to vulnerabilities such as tampering, information disclosure and denial of service. Only data flows that cross process and trust boundaries will be considered here. The purpose of this consideration is to reduce the number of

irrelevant elements in this analysis. It is clear that there are several ways the data flow to or from a controller which could be exploited by an intruder to compromise the security of a controller. From the simplified data flow model in Figure 5.3 data flows are through three interfaces such as SBIs, NBIs and Administrative interfaces. These three interfaces are analyzed below as data flows.

- **SBIs**

In the context of SDN, data flows represent the actual transmission of data between the switch and the controller. However, data flows are also the most vulnerable elements susceptible to attacks. Data flows in SDN controllers include PACKET IN, FLOW MOD, and FLOW REMOVED messages. To address security concerns, it is possible to prevent information disclosure and tampering in the Ryu SBI [77] by establishing and configuring a TLS or SSL connection. The rate of traffic to the controller is directly related to the number of packets that do not match the current flow rules. If the network capacity is exceeded and congestion occurs, the switch's ability to react to new flows will be negatively affected. Significantly, there is latency between locally handled packets on the switch and those that need to be transmitted to the controller. This latency can lead to information disclosure, enabling the determination of whether a packet matches a flow rule and identifying the installation of new flow rules through subsequent packet delivery. If an attacker has direct access to the management network, further issues can arise. In this scenario, it is possible to conduct traffic analysis of the communication between the switch and the controller. This analysis provides more detailed information than time analysis alone, as it allows determination of packet sizes and the controller's response. Access to the management network opens up additional opportunities for traditional TCP/IP attacks that can interfere with the communication between the switch and the controller, potentially leading to denial-of-service attacks. Fortunately, Ryu integrates the Snort intrusion detection and prevention module [78], either as an overlay or standalone, to enhance security for these data flows. With the integration of Snort, the data flows within Ryu can be protected against denial-of-service attacks.

- **NBIs**

The communication between the controller and applications, or vice versa, is established through NBIs. In this context, data flows represent the transmission of various types of data, such as application state, user session information, and data required by the controller for managing applications. Ryu follows a monolithic architecture, where the main process runs as a single

entity, and each application and service operates on its own thread within the same process context. This architecture allows applications to subscribe to relevant events of interest. When an event occurs, the event handler thread distributes it to the subscribed applications accordingly. It is also possible for an application to subscribe to multiple events. However, it is important to note that Ryu lacks an authentication method in the NBI, making it susceptible to spoofing attacks. This vulnerability exposes the data flow to various types of attacks, including data manipulation. An attacker can manipulate the data flows between application layer components and the controller by inserting malicious code, altering data, or tampering with packets. Such attacks can lead to data corruption, denial of service, and unauthorized access to information. Another concern is the lack of encryption and the use of HTTP instead of HTTPS in the NBI channel. This opens up the possibility of disclosure in the data flow between the controller and northbound applications. Attackers, especially "Man-In-The-Middle" (MITM) actors, can compromise this data flow. By employing simple techniques like ARP spoofing and MITM attacks, hackers can intercept and obtain all the information exchanged between the controller and the northbound application. Additionally, when the communication channel is not secured, an intruder can modify the content of packets, further demonstrating the susceptibility of the NBI to data manipulation and tampering.

- **Administration Interface**

It is possible to attack the system through the controller's administration interface. In most of network systems, this interface is strongly secured. It will be vulnerable to attack, if the administrator accidentally or intentionally lost or transfers his credential to unauthorized person or makes accidental errors which make the system vulnerable to attack when he/she configures the system. Assume that all goes well and this interface of the controller is secured from all potential vulnerabilities and then we exclude it from this vulnerability analysis.

➢ **Data stores**

Here, data stores of the controller are internal data structures like databases, memory locations for logs, policies and the like. As mentioned in our scope, this study doesn’t include the security analysis of controller internal implementation details. So the analysis on data stores is not applicable here.

- **Process**

In this analysis, we will focus on the controller core process. Like any other process, the controller core process is susceptible to various security issues covered by the STRIDE methodology, as mentioned in Section 3.2.1. One notable security concern in the Ryu controller is the absence of an authentication method in both NBIs and SBIs. As a result, the controller core process is highly vulnerable to spoofing attacks. Moreover, it is prone to data manipulation. An intruder can easily tamper with flow tables on a switch by sending PACKET OUT and FLOW MOD messages, taking advantage of the fact that the controller runs user programs within a process context. Prior to sending these messages to the switch, the Ryu controller lacks the capability to verify whether a user or application is authorized to perform such actions. A denial of service (DoS) attack is also feasible in the Ryu controller core. By executing the Ryu's exit (1) command, which triggers an infinite loop, an attacker can initiate a DoS attack from any thread, rendering the system unresponsive. This vulnerability arises from the fact that access privileges of applications are not properly checked, allowing application threads to excessively consume controller resources, leading to system disablement. Additionally, there is a likelihood of privilege escalation occurring within the Ryu controller core. Furthermore, the absence of authentication and logging mechanisms in the NBI of the Ryu controller creates the potential for repudiation attacks on the identity of NBI applications. However, this issue has been addressed in the SBI, where a logging system captures the data flow ID of each received message [57].

- **Interactors**

Switches in the infrastructure layer and applications in the application layer are the controller system's interactors. As previously mentioned in our scope, this study covers only SDN controllers and their communication interfaces, thus interactors of the controller are out of our scope of this analysis.

As a result, the following threats in Table 5.1 have gotten in Ryu controller. Hence, we will summarize our analysis by using STRIDE threat matrix. In this matrix "✓" indicates that mitigation mechanism exist for that attacks on vulnerabilities, "X" indicates the possible existence of a threat but no mitigation mechanism exist, "-" indicates no information is provided for that threat, blank space indicates component cannot be affected by threat.

Table 5.1 Ryu controller STRIDE threat matrix

Type	Components	S	T	R	I	D	E
Data Flow	NBI		X		X	-	
	SBI		✓		✓	✓	
Process	Controller Core	X	X	-	-	X	X
Data Store	Internal data structure		-		-	-	
Interactors	Switches and Applications	X		X			

- **POX controller**
- **Data Flows**
- **SBIs**

To ensure secure communication between switches and the POX controller, the POX controller utilizes the TLS protocol. Furthermore, it incorporates IEEE 802.1X and RADIUS authentication server [82] based AuthFlow [83] for enhanced security. These advanced techniques are employed to authenticate and secure both the SBIs and NBIs communications. When a request is sent to the controller, AuthFlow forwards it to the authentication server, which then verifies the credentials' validity using RADIUS authentication. Upon successful authentication, the authenticator employs SSL and Public Key Infrastructure (PKI) to send a confirmation message to the POX controller. AuthFlow utilizes the authentication credentials to enforce access control based on the privilege level, ensuring that data tampering is impossible on SBIs due to the authenticated and guarded nature of every communication. While there is a potential for information disclosure in POX SBIs, this risk can be mitigated by implementing connection request time synchronization [83]. To address denial-of-service (DoS) attacks in SBIs, rate limiting and packet dropping can be utilized as effective mitigation strategies within the POX controller's SBI component.

- **NBIs**

To ensure the security of communication between the controller and northbound applications over NBIs, HTTPS is utilized. HTTPS protects against channel tampering by encrypting the

communication. Additionally, application layer authentication of each user or application in the NBI further strengthens the security of this communication interface, guarding it against Denial-of-Service (DoS) attacks. Similar to SBIs, the NBI also employs the connection request time synchronization method [83]. This synchronization method prevents information disclosure in the NBI interface, ensuring that sensitive data remains protected.

- **Administrative Interface**

Similar to Ryu controller, the administration interface of POX controller is assumed to be secured that is in this interface all goes well and it is free from potential vulnerabilities. So it is not included in the security analysis.

➢ **Data Stores**

Data stores are in the internal implementation of the controller, they are not considered in this study based on the defined scope of this study.

➢ **Processes**

In this analysis, we will focus on the security of the core process of the POX controller. The POX controller has implemented a strong authentication system and does not allow unauthorized access. This authentication method can detect invalid ARP header fields, preventing spoofing of the controller core. However, there is a possibility of information disclosure in the POX core. When a new rule is added to the connection request, the controller core may disclose information based on the timing of TCP setup. This behavior is observed in the forwarding.l3 aggregator simple module of the POX controller. Since the switches and northbound applications communicate with the POX controller, the controller core does not keep logs. This lack of logging can potentially enable repudiation within the controller core. A denial-of-service attack can be launched by sending a large number of forged packets to the controller, causing the flow table to overflow as each packet create a new flow rule. This attack can be carried out using the forwarding.l2 learning module [71], potentially overloading the network controller. To address the threat of elevated privileges, the AuthFlow [83] mechanism is employed. It restricts access to the control based on the privilege level of the POX processes.

➢ **Interactors**

In similar way for the Ryu, interactors such as switches and SDN applications are not included in our analysis.

From the STRIDE threat analysis, we have found threats in POX controller, thus we will summarize our analysis by using the following STRIDE threat matrix shown in Table 5.2.

Table 5.2 POX controller STRIDE threat matrix

Type	Component	S	T	R	I	D	E
Data flow	NBI		✓		✓	✓	
Data flow	SBI		✓		✓	✓	
Process	Controller core	✓	✓	X	-	-	✓
Data store	Internal data structures		✓		✓	-	
Interactors	Switches and Applications	-		X			

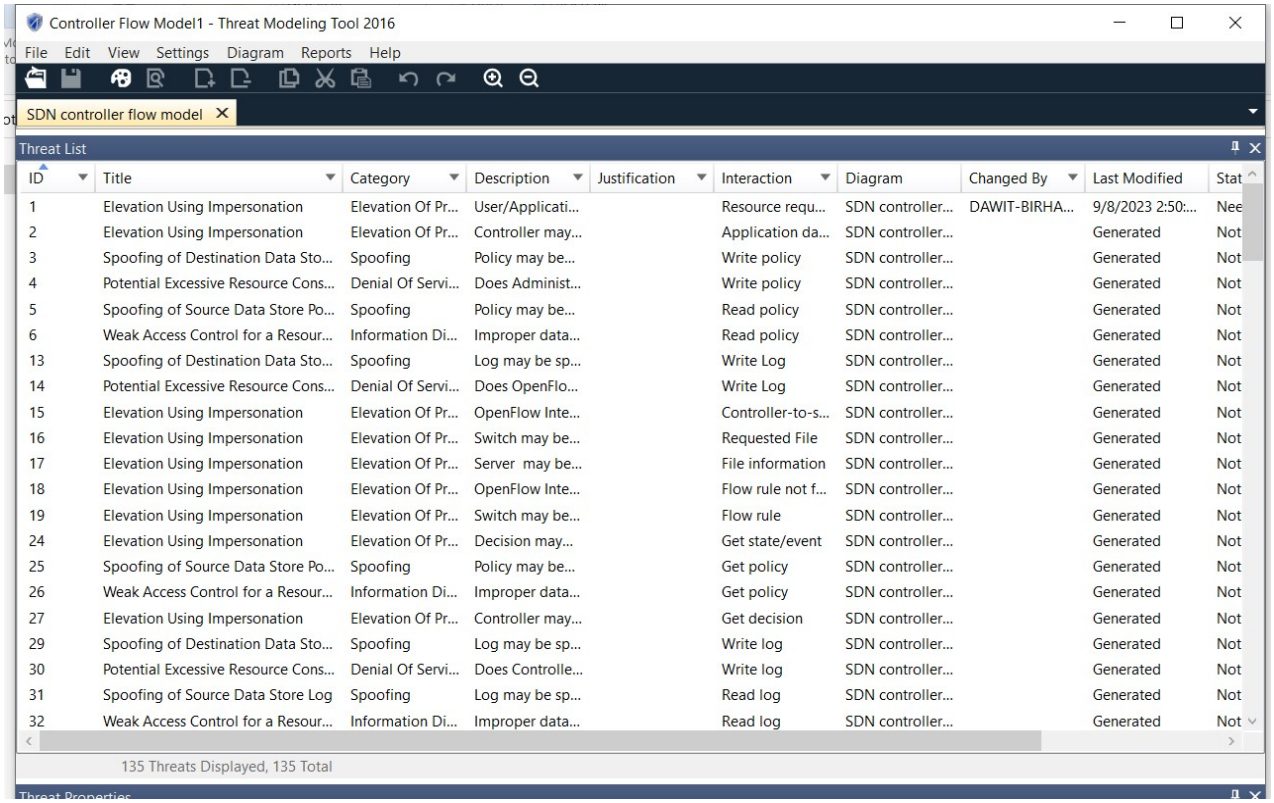


Figure 5.4 STRIDE threat analysis view

Based on the STRIDE threat model, vulnerabilities on both controllers are identified and classified as shown in Figure 5.4 and summarized in Table 5.1 and Table 5.2 for Ryu and POX controllers respectively.

5.2.3 Attack Tree Threat Model Security Analysis

The threat vectors that could compromise the controller were identified in the previous section. In this section, we clearly outline how threats or attacks are carried out from the viewpoint of an intruder. Attack trees, which are a graphical depiction of attacks in the form of a tree data structure, are described in Chapter 2. The attacker's ultimate objective is represented by the tree's root in this structure, while the attacker's various attack methods are represented by the nodes attached to the root. To execute the assault or achieve the attacker's final objective, the OR and AND operators define the logical linkages between the tree branches. Now, we use the component and vulnerabilities pair we have gotten from the data flow and STRIDE models as an input to attack trees and by using Algorithm 4.3 to explore how the identified vulnerabilities can be exploited by an attacker and analyze the specific attack steps and attack paths for each threat to analyze threats that can be uncover by STRIDE in depth and in detail. Let us use these resulting potential vulnerabilities and controller component pair to present the attack tree analysis by using attack tree threat model.

- **Spoofing**

This attack consists of attempting to fool the controller that an attacker is somebody who is legitimate user or trusted application other than who he/she/it actually is. It is possible by hacking identity, which is not exclusive to individuals but also machine identity (IP or Media Access Control (MAC) Addresses) and processes running on a host and even file may be spoofed. To spoof a MAC address or an IP address; it is also possible to send forged ARP and/or router advertisement packets to attempt to solicit traffic destined for other hosts. The controller components are exposed to various spoofing threats. The attack tree is shown below:

Goal: Spoofing access to the controller; OR

Spoofing authentication; AND

Spoofing the username; OR

Social engineering; OR

Brute force attacks

Spoofing the password

Social engineering; OR

Brute force attacks

Spoofing the source address

Spoofing MAC address; OR

Spoofing IP address

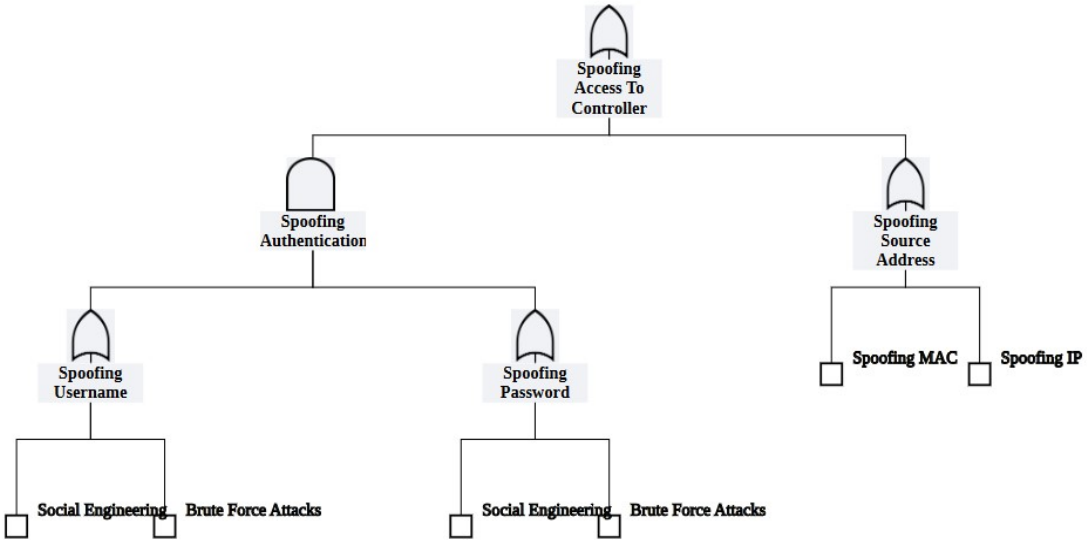


Figure 5.5 Attack tree for spoofing attack on the controller

- **Tampering**

A tampering attack is an attempt to modify or delete data to falsify information in the controller core or data flows and trick the controller to install flow rules which may perform some packet modification. This can be done by exploiting vulnerabilities in the controller software, or by gaining unauthorized access to the controller. The goal of a tampering attack is to disrupt the operation of the network or to steal sensitive data. The following attack tree represents tampering attack to the controller core and the data flows to the controller.

Goal: Tampering data; OR

Tampering data in the data flows (NBIs); OR

Injecting data; OR

Deleting data; OR

Modifying data; OR

Replaying data

Tampering data in the controller core

Injecting data; OR

Deleting data; OR

Modifying data; OR

Replaying data; OR

Injecting code

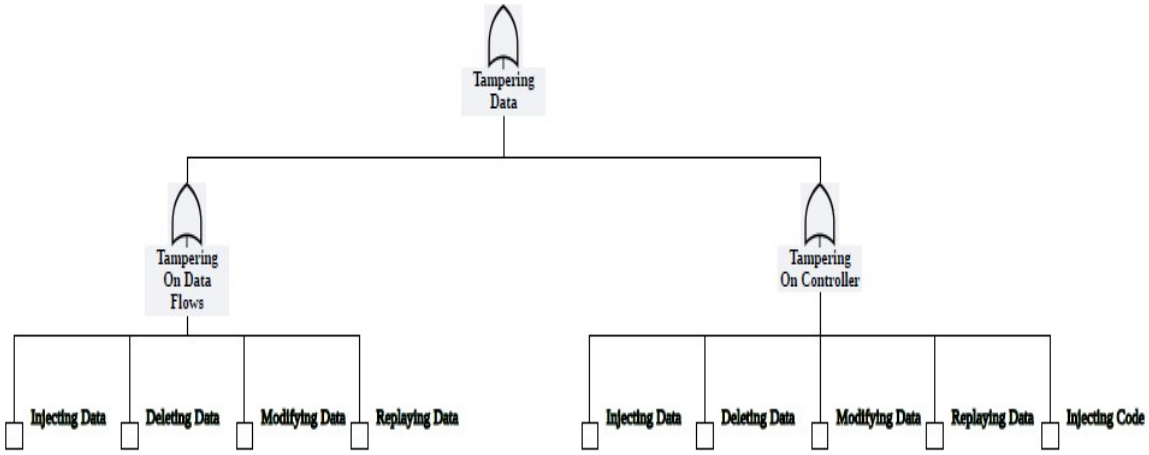


Figure 5.6 Attack tree for tampering attack on controller dataflow and core

✓ **Repudiation**

It is an attempt to deny responsibility for an action that was taken. In the context of our study area a repudiation attack could be used to deny responsibility for sending a packet, or for making a change to the network configuration. An attacker may forge source addresses for packets, therefore there is no assumption that packets genuinely originate where the packet headers indicate. The attack tree of this attack is shown below:

Goal: Repudiate interactors of the controller; OR

Spoofing the identity; OR

Spoofing the identity of the sender of the packet; OR

Spoofing Media Access Control (MAC) Address; OR

Spoofing IP Address

Spoofing the identity of the receiver of the packet; OR

Spoofing Media Access Control (MAC) Address; OR

Spoofing IP Address

Modifying the contents of a packet; OR

Intercepting a packet to replay later; OR

Man-in-the-middle

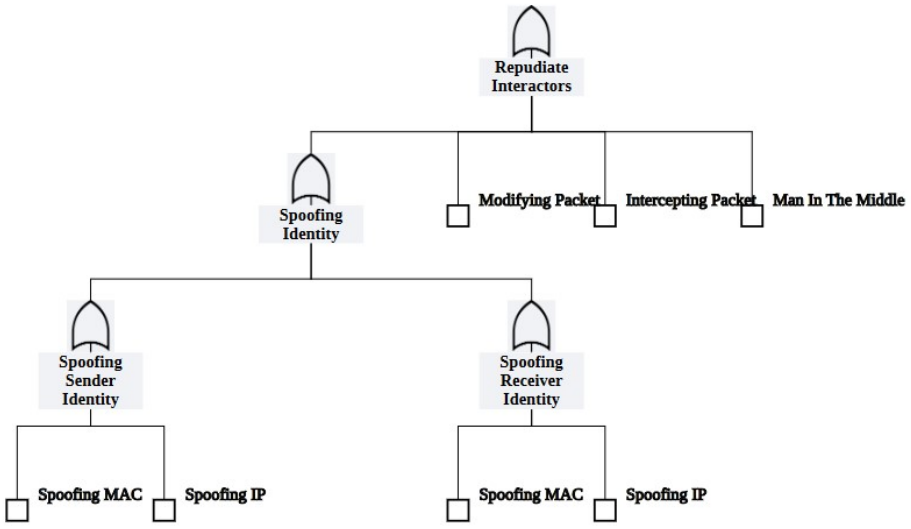


Figure 5.7 Attack tree for repudiation attack on controller interactors

✓ **Information disclosure**

An information disclosure attack on an SDN controller is an attempt to gain access to sensitive data, such as the network topology, the traffic patterns, or the credentials of network administrators. This can be done by exploiting vulnerabilities in the controller software, data flows or by gaining unauthorized access to the controller. The attack tree is shown below:

Goal: Disclosing controller data; OR

Sniffing the communication channels; OR

Poisoning Address Resolution Protocol (ARP)/ Man-in-the-Middle (MIME); OR

Spoofing Domain Name System (DNS); OR

Spoofing Internet Protocol (IP)

Analyzing traffic; OR

Sniffing packet

Accessing controller; OR

Exploiting vulnerabilities; OR

Accessing physically the machine where the controller software resides in

Bypassing authentication; OR

Social engineering; OR

Brute force

Attacking controller web interactions

Injecting SQL; OR

Cross-site scripting

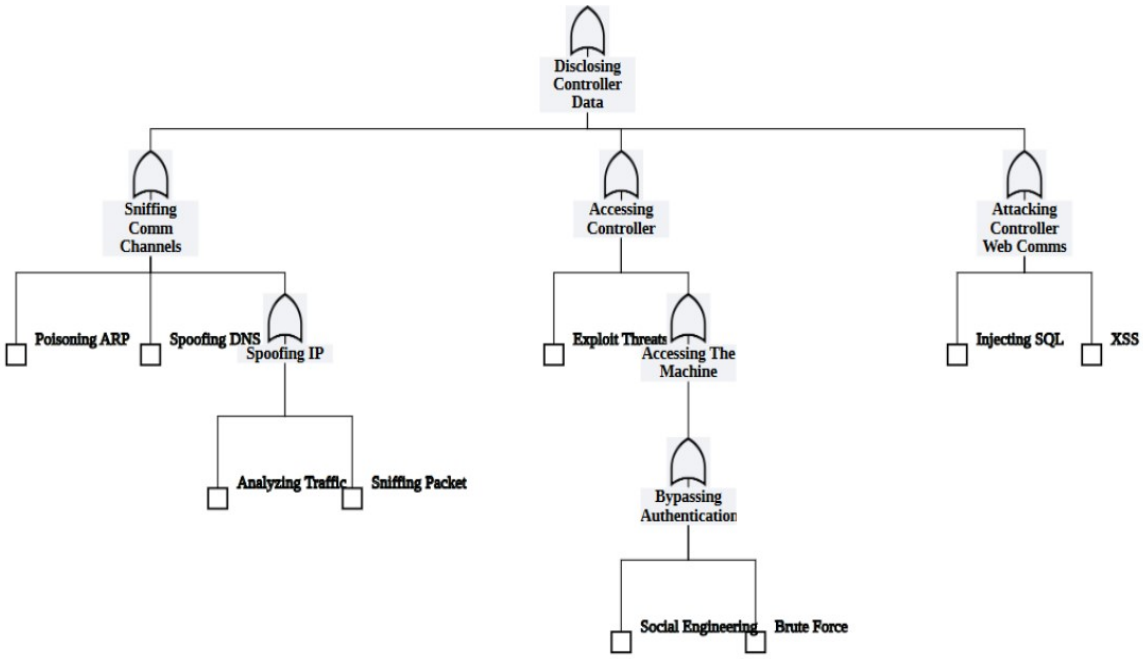


Figure 5.8 Attack tree for information disclosure attack on controller

✓ **Denial of Service**

A denial-of-service (DoS) attack on an SDN controller services is an attempt to make the controller unavailable to legitimate users. This can be done by flooding the controller with traffic, sending it invalid requests, or exploiting vulnerabilities in the controller software.

This attack is the security risk where controller provides the largest attack surface. The fact that packets must be sent to the controller on a regular basis opens up a number of potential routes for denial of service attacks.

Goal: Denial of controller services; OR

Flooding requests to controller services; OR

Interrupting controller services; OR

Exhausting controller state

Flooding service abstraction layer buffer; OR

Exhausting resources

Congesting network bandwidth

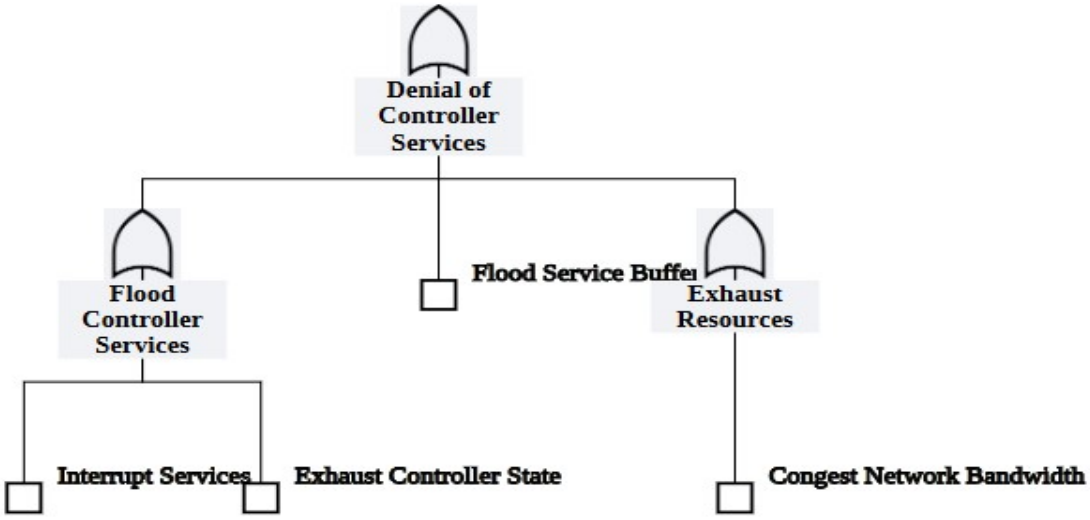


Figure 5 9 Attack tree for DoS attack on controller services

✓ **Elevation of privilege**

An elevation of privilege attack (EoP) on an SDN controller is an attempt by an attacker to gain unauthorized access to the controller or to its resources.

These attacks are based on program flaws, employing techniques such as fuzzing, static analysis and reverse engineering to reveal coding deficiencies. Intruders without privileges can exploit vulnerabilities to execute remote code; subsequently, in the post-exploitation phase, they can escalate their privileges to break into the system entirely. The SDN controller software is likely to have coding flaws, which may be exploited for privilege escalation attacks. The attack tree is shown below:

Goal: Elevating privilege for unauthorized access to the controller; OR

Exploiting vulnerabilities in the controller; OR

Exploiting buffer overflow vulnerabilities; OR

Exploiting insecure permissions; OR

Exploiting misconfigurations

Analyzing controller code; OR

Analyzing controller code statically; OR

Analyzing controller code dynamically

Elevating standard privilege to higher privilege on access to the controller

Bypassing authentication

Social engineering

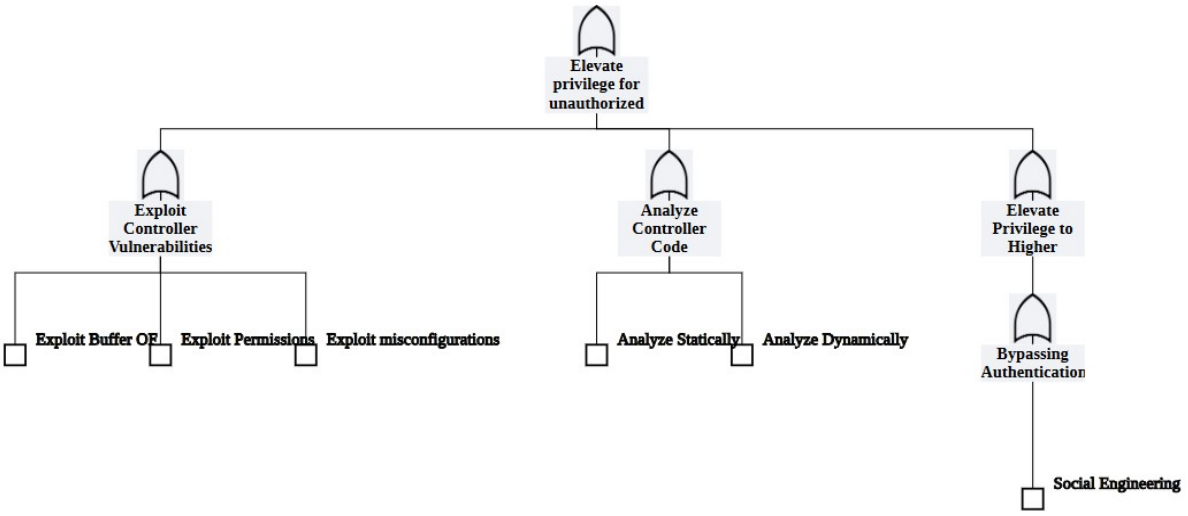


Figure 5.10 Attack tree for elevation of privilege attack on the controller core

Now, let us summarize the attack types found as a result of our analysis so far by using our hybrid threat model. Such attack types found as end result of attack tree analysis by which attackers can exploit it using the potential vulnerabilities identified and classified by STRIDE on the controllers' components and data flows. They are summarized and presented in Figure 5.11 and Figure 5.12.

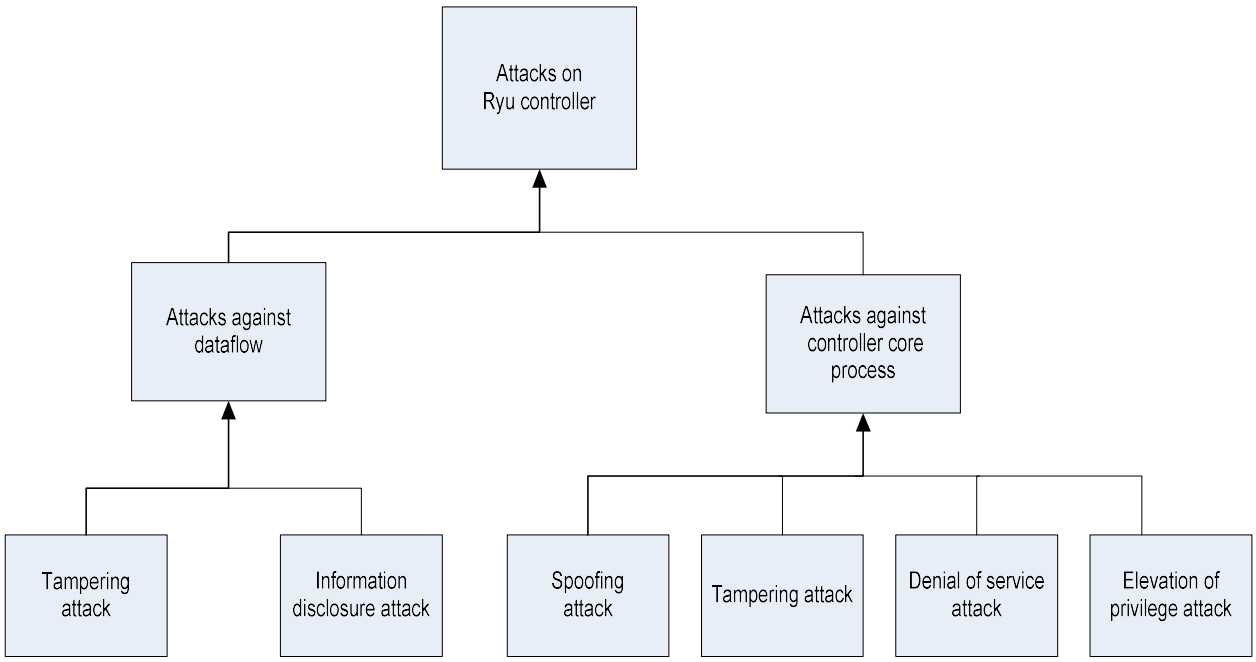


Figure 5.11 Attacks against Ryu controller summary analysis result of hybrid model

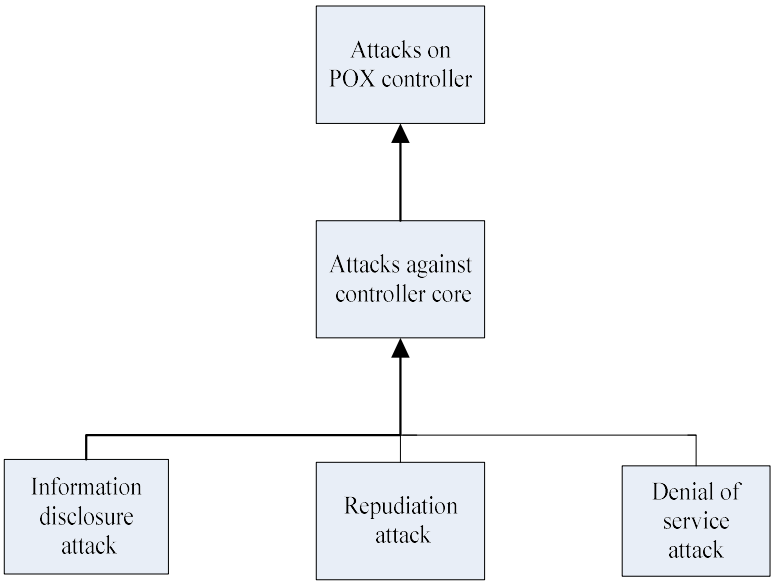


Figure 5.12 Attacks against POX controller summary analysis result of hybrid model

The result summary shown in Figure 5.12 of POX controller clearly indicates that analysis result of our proposed hybrid threat model is different from the STRIDE only model. By using our proposed hybrid model in addition to vulnerability to repudiation attack which is found by STRIDE only model, we have identified vulnerabilities to information disclosure and DoS attacks on POX controller core. Table 5.3 presented STRIDE matrix of analysis result of our proposed hybrid threat model.

Table 5.3 STRIDE threat matrix of proposed hybrid threat model for POX controller

Type	Component	S	T	R	I	D	E
Data flow	NBI		✓		✓	✓	
Data flow	SBI		✓		✓	✓	
Process	Controller core	✓	✓	X	X	X	✓
Data store	Internal data structures		✓		✓	-	
Interactors	Switches and Applications	-		-			

5.2.4 Security analysis result comparison

In this section, we will present our hybrid threat model security analysis results in comparison with the results from STRIDE only model used in previous studies. To evaluate our proposed hybrid threat model analysis results and compare with STRIDE only threat model, we have defined two scenarios as:

Scenario 1, the STRIDE only analysis identified information disclosure vulnerability in the data flows of the Ryu controller. By incorporating attack trees into the analysis, the hybrid model provided a more detailed understanding of the attack vectors and their potential consequences. It revealed that the information disclosure vulnerability could be exploited through packet sniffing and protocol analysis, leading to the extraction of sensitive data and compromising the system's security. The hybrid model enhanced the effectiveness of the STRIDE only analysis by providing a more detailed understanding of the specific attack vectors and their potential impact.

Scenario 2, the STRIDE only analysis identified vulnerabilities of tampering, denial of service, and elevation of privilege in the controller core of the Ryu controller, and repudiation vulnerability in the POX controller core. The hybrid model uncovered that a combination of code injection and privilege escalation could allow an attacker to modify critical control logic within the controller, resulting in unauthorized control over the network infrastructure or complete system compromise. Additionally, the hybrid model detected information disclosure and denial of service vulnerabilities in the POX controller core. By comparing the results of the STRIDE only analysis with the hybrid model as shown in Table 5.4, we demonstrated how the attack trees provided a more detailed understanding of potential attack paths and associated risks for our proposed hybrid model. Overall, the hybrid model offers a comprehensive and detailed analysis of vulnerabilities and potential attacks on the controllers. By leveraging the structured approach of STRIDE and the graphical representation of attack trees, the hybrid method enhances the analysis and provides a clearer understanding of the security risks involved.

Table 5.4 STRIDE category threat detection comparison of STRIDE only and proposed hybrid threat models

S.No.	Controllers		STRIDE only model	Proposed Hybrid Model
			STRIDE category threats found	**STRIDE category threats found**
1	**Ryu**	**Data flow**	Tampering, information disclosure	Tampering, information disclosure
		Core process	Spoofing, Tampering, Dos , elevation of privilege	Spoofing, Tampering, DoS, elevation of privilege
2	**POX**	**Data flow**		
		Core process	Repudiation	Information disclosure, Repudiation, DoS

In summary, our thesis work aimed to overcome the limitations of the STRIDE only threat model by designing a hybrid threat model that combines STRIDE and attack tree models. The hybrid model provides a more comprehensive and detailed analysis of security threats in SDN controllers, as illustrated by the scenarios we evaluated above. Our proposed model has shown improved effectiveness in detecting vulnerabilities compared to the STRIDE only model by identifying information disclosure and DoS attack vulnerabilities on POX controller core.

5.3 Summary

The proposed hybrid threat model has been presented and its operation has been explained in chapter 4. In this chapter, the tools used to implement the proposed hybrid threat model and then the implementation of the method for security analysis of RYU and POX SDN controllers. Several security flaws have been found and categorized by using STRIDE threat categories, and after that the various ways an attacker could exploit those flaws and attacks the controller have been modeled. Analysis of the security of RYU and POX using the proposed hybrid threat model can provide valuable insights into potential security risks and vulnerabilities. Here's a summary of such analysis:

As we have shown from the analysis there is a potential for unauthorized access to RYU controller components and its communication channels with other layers which leads to spoofing attacks. Strong authentication and access control is essential to prevent it. It is also vulnerable to data tampering on its data flows. Ensuring message integrity and encryption can mitigate this risk. Explored scenarios where RYU controller might fail to log important events leading to repudiation attacks. Robust event logging and auditing mechanisms are needed to tackle this attack. Analysis also indicates that there is a risk of sensitive data leakage during controller communication. So, encryption and proper access controls are essential to prevent disclosure of information. There is also a potential of DoS attacks against it including resource exhaustion. Implementing rate limiting and anomaly detection can help to mitigate this attack. During security examination of this controller, there are also scenarios where attackers might escalate their privileges within RYU components. Strict access control and regular security updates are necessary as prevention. According to our analysis, relatively POX controller is more secure than RYU controller, but it has security vulnerabilities that exploit it to information disclosure, DoS and repudiation attacks. In the previous study in [25], the security analysis of 4 commonly used SDN controllers were studied by using STRIDE only method, from those four controllers POX was the one in their study. According to their analysis result, the POX core is not vulnerable to DoS attack but in our analysis by using our proposed hybrid threat model, we can found that POX core is vulnerable to DoS attack too. This finding of our analysis will be evaluated in our experimental testing of exploitation of this attack on POX controller. To mitigate the attacks on POX controller, similar measures we listed above for RYU controller should be applied.

Chapter 6: Experimental Evaluation of the Proposed Hybrid Threat Model

In this chapter, exploit for at least one of the security issues detected in Chapter 5 will be developed. This Chapter further divided into the following sections: Section 6.1 examines the different issues regarding exploitability, Section 6.2 describes the simulation setup and Section 6.3 describes the execution and presents and discussed the results of the attempted exploitation.

6.1 Selection of security vulnerabilities for exploitation

Due to different factors listed below here, execution of all security issues found in section 5.2 is impossible. Therefore, we should select at least one security issue to demonstrate and evaluate the results of the proposed hybrid threat model experimentally. In order to choose a vulnerability to exploit, it is necessary to define certain criteria. Firstly, the vulnerability should be practical to exploit in virtual environment, that is, with the available tools it should be possible to perform the operation in a reasonable period of time. Secondly, the vulnerability should be reliably exploitable, that is, it should be possible to perform the exploitation consistently, without relying too much on factors that we have no control over. Thirdly, the exploitation should be possible, or feasible with SDN controllers. Finally, the attack should have notable consequences. Exploitation of attacks is illegal task, due to this reason, there is a deficiency of resources to implement all such attack types on the investigated vulnerabilities of the controller. Our security analysis result for POX controller, differs from the result of previous studies by using STRIDE only method on DoS vulnerability. Based on these criteria and it is commonality in both of the controllers under this study, denial of service (DoS) attack is selected to demonstrate and evaluate the results of the proposed hybrid threat modeling method. Denial of service is an attack based on generating a large amount of traffic towards the controller in order to overload some aspect of the system. It is the easiest to exploit, have a substantial impact on the controller system and even on the whole SDN.

6.2 Simulation Setup

6.2.1 Hardware Environment Setup

Security analysis of controllers were performed in an environment of a Windows 10 Pro laptop computer with Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz and 8.00 GB RAM specification.

6.2.2 Virtualization Environment Setup

Oracle VirtualBox 7.0.8 is used as the hypervisor to instantiate two Virtual Machines (VMs), each for one type of controller. Each VM is allocated 1 CPU and 2 GB of RAM. All the simulations run on Ubuntu Desktop 20.04.1. Each VM contains Mininet with installed Ryu or POX controller. Each controller run on an individual VM and connected to predefined custom topology. The IP address used for both controllers is 127.0.0.1 and their listening port numbers assigned to Ryu and POX controller are 6653 and 6633 respectively. Python 2 and 3 are interchangeably used as a scripting language to write our custom network topology and for connecting and communicating controllers.

- **Network Emulator**

Mininet is a popular open-source network emulator that allows us to create virtual networks on a single machine. It provides us a lightweight environment for developing, testing, and experimenting with SDN. We can quickly create our custom network topology comprising virtual switches, hosts, and controllers, and simulate our network behavior in a controlled manner. It is a suitable environment to simulate network topologies and experiment with different network configurations. The "mn" command in Mininet is used to create various network topologies with different characteristics. The "mn" command can create a simple single switch topology, linear topology, tree topology and also allows us to create custom topologies by specifying the desired network configuration using python code. This flexibility enables us to create complex network topologies with specific requirements or to emulate real-world network scenarios. In this research we have used mininet 2.2.2 and define our own custom topology by using miniedit and python script saved as “my_custom_topology.py” its code is documented in Annex A - my_custom_topology.py script code. Miniedit is a graphical user interface (GUI) tool that provides a visual way to create and customize network topologies in Mininet. It allows us to design and manipulate network topologies by dragging and dropping controllers, switches, hosts, and links onto a canvas. Miniedit simplifies the process of creating complex network topologies by providing an intuitive interface that abstracts the underlying code required to define the topology. By using this tool we can easily add, remove, or modify network elements and their connections, making it convenient for network experimentation, testing, and educational purposes. We have created our SDN network topology shown in Figure 6.1 for our study by using miniedit GUI.

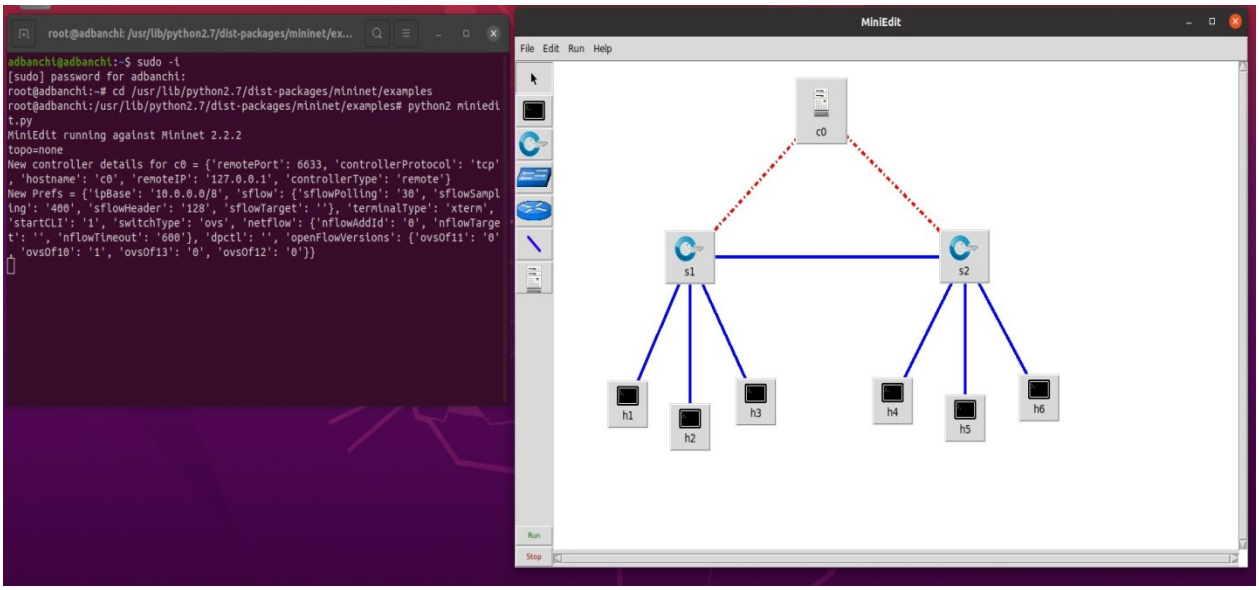


Figure 6.1 Custom topology created by miniedit

➢ **Ryu and POX Controller**s

As we mentioned in Section 2.2.1 of this thesis document, there are different types of free and licensed OpenFlow controllers. Ryu and POX SDN controllers are open source controllers which use apache free license. For this experiment, we downloaded and configured these controllers software in our Ubuntu machine. Controllers are implemented by python language, this makes seamless integration with the mininet emulator and also it increases their ease of use. Due to this seamless integration most of the time POX controller is incorporated inside the mininet software. For this research work we use both controllers as remote controllers, so we download and install each of them in their separate VMs outside the mininet software. Commands used to install these controllers and mininet emulator is listed in Annex B – Mininet, POX and Ryu SDN controllers' installation commands. Figure 6.2 shows the POX and Ryu controllers' startup by using their startup commands of the controllers: *python3 ./pox.py forwardingl2_learning* is the startup command to the POX controller and *ryu-manager ryu/ryu/app/simple_switch_13.py* is startup command to the Ryu controller. Figure 6.3 and Figure 6.4 show respective connection of the controllers for remote network control responsibility after the controllers' setup has finished and our custom topology has ran.

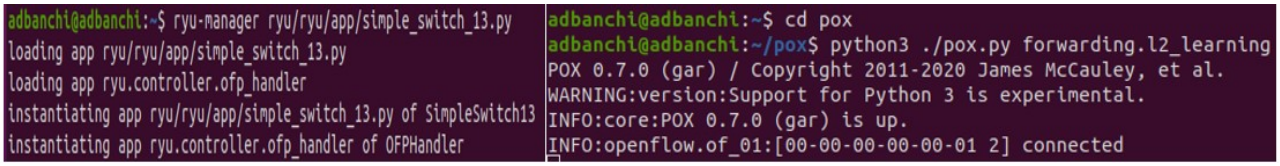


Figure 6.2 Startup of Ryu and POX controllers by using their start up commands

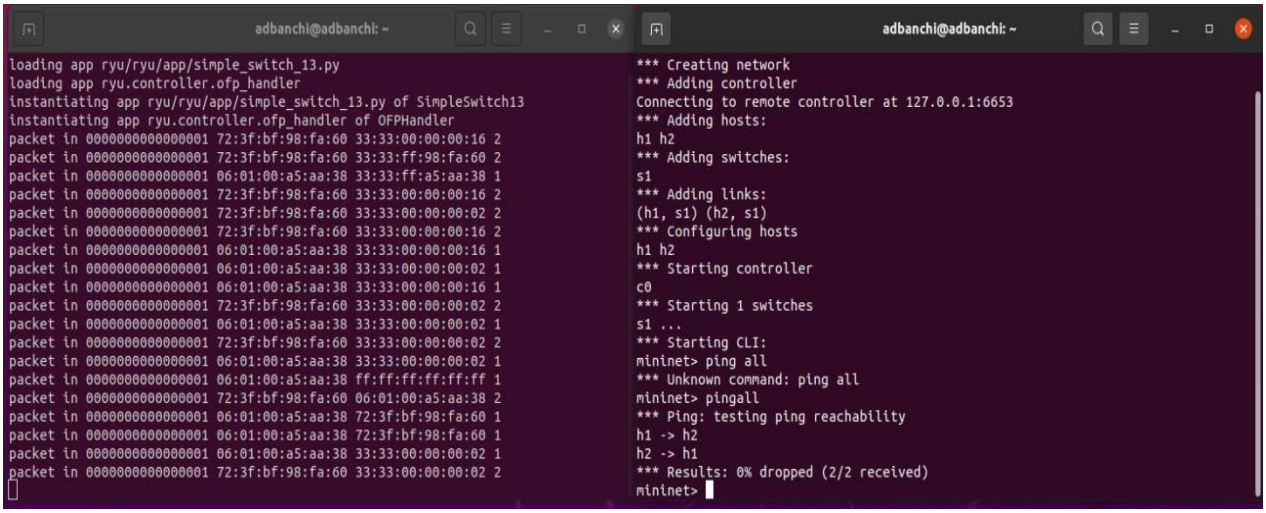


Figure 6.3 Ryu controller connection

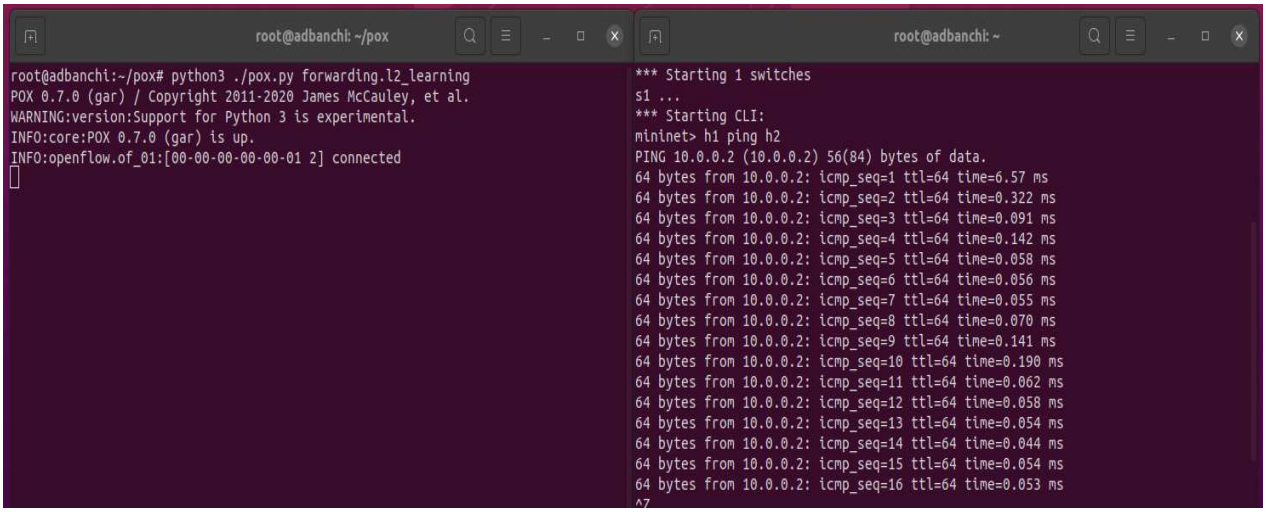


Figure 6. 4 POX controller connection

✓ **Traffic Generation, Monitoring and Testing Tools**

✓ **Ping Tool**

A ping tool is a network utility that is commonly used to test the reachability and latency of a network host. While ping is primarily used for network diagnostics, it can also be utilized as a basic performance testing tool. A performance testing feature that a ping tool can offer is a measure of round-trip time (RTT) between sender and receiver. This provides an indication of network latency, which is the time it takes for a packet to travel from the source to the destination and back. Latency measurement can help us to identify potential delays in network

communication. In addition to this it can detect packet loss by examining the responses received from the target host. When a packet is lost, the tool can calculate the packet loss rate, indicating the percentage of packets that did not reach their destination. These features of ping tool are important and used in our DoS attack exploitation on the controllers to indicate the attack impact on their performance. The commands- *pingall* or *h1 ping h2* (where h1 and h2 are hosts we want to check their connectivity) are used to use this tool.

- **Iperf Tool**

Iperf [28] is a widely used open-source tool for measuring network performance. In order to access the impact of DoS attack on the performance of POX and Ryu SDN controllers we have been used it as a performance measuring tool. It allows us to assess the performance parameter such as network bandwidth by generating traffic between a client and a server architecture created between the hosts in our custom SDN topology and use it to measure the maximum achievable bandwidth between two network endpoints before and after DoS attack is launched. We have downloaded and installed it on Ubuntu, its installation is shown in Annex B. the command used to install it is - *sudo apt-get install iperf* . After installing iperf then start iperf server and client with the following commands: To start the iperf` server, we open a terminal or xterm window of the server host and run the - *iperf –s* command including optional values of the port number and the time interval between packets. After it starts it waits for incoming connections. To start and run the iperf` client for TCP/UDP traffic on the client host, open its terminal or xterm window and run the command- "*iperf -c <server_ip_address>"* command including optional values of the port of the server the time interval for the request etc. By default, iperf` will use TCP as the transport protocol. Finally the client will connect to the server, initiate the TCP session, and start sending data. To run the `iperf` client for UDP traffic and to generate the traffic, we need to specify the *`-u`* flag when running the `iperf` client. On the client machine, terminal or xterm window, we run the command – "*iperf -u -c <server_ip_address>"*, here the `-u` flag instructs `iperf` to use UDP as the transport protocol. In similar way of TCP traffic the client will connect to the server and start sending UDP packets. After communication starts between two hosts, both the client and server machines iperf` will display the results of the test such as network bandwidth and other statistics. The client can indicate the amount of data sent and the measured performance metrics. By following these steps, we have used iperf` to generate normal TCP and UDP traffic between a client and server. The tool allows us to assess network performance by measuring bandwidth and latency and then by using the number of packets passed through the controller we

can evaluate the quality of the connection in relative to the DoS traffic generated by using hping3 tool.

- **Hping3 Tool**

It is a powerful network packet crafting and manipulation tool that can be used for various purposes, including network testing, analysis, and security auditing [29]. Hping3 allows us to construct and send custom packets with specific characteristics including size, quantity, and fragmentation of packets in order to overload the target. These features make it useful for SDN controllers' DoS exploration in our experiment. It is mostly used in security testing, so we will test controllers' vulnerability to DoS attack by establishing a big amount of connections in our network by using this tool. Before starting hping3 for our security testing, we have installed it on Ubuntu with the command-" *sudo apt install –y hping3".* Its installation is documented in Annex-B. After installed it on our OS, then we have started the iperf server with the iperf command and then start the iperf client (it represents the attacker) with the hping3 command-" *hping3 –c <number of packets> -d <byte size> -S –w<window size> -p<port number> --flood --rand-source <destination IP address>".* This command is used for TCP SYN flood type DoS attack; we will add or remove the parameters and use it as DoS implementation tool based on our requirements.

- **Wireshark Tool**

Wireshark is an open-source tool widely used as network protocol analyzer [31]. It allows us to capture and inspect network traffic in real-time and provides detailed visibility into the packets flowing through a network. With this tool, we can capture network packets from various interfaces such as Ethernet, Wi-Fi, and loopback. It also provides a graphical interface that allows us to view captured packets, analyze their contents, and extract valuable information. Wireshark supports a wide range of protocols, including TCP/IP, UDP, HTTP, DNS, and many others. In this work we generate and used only TCP and UDP traffics which use TCP/IP and UDP protocols, This tool in its detail decodes and displays the information contained in each packet, like source and destination IP addresses, port numbers, protocol headers, and payload data. This tool offers powerful filtering and search capabilities, enabling us to focus on specific packets or network conversations of interest. We also can apply filters based on various criteria, such as source or destination IP addresses, protocols, port numbers, and packet contents. Wireshark also provides features like packet coloring, packet time stamping, packet statistics, and the ability to

export captured data for further analysis or sharing with others. Overall, this tool is a valuable tool for network security analysis by giving understanding about network behavior. Its user-friendly interface and extensive protocol support make it a suitable tool for network us. In our network setup after starting up our controller and our network topology, we opened the xterm window of the hosts we want to capture its traffic and then start our preinstalled Wireshark tool on our Ubuntu OS by using the following command- "*sudo wireshark &".*

6.3 Experimentation and Results

To evaluate our proposed hybrid model, we experimentally test and demonstrate the exploitation and impact analysis of DoS attack which is one of security analysis result of our proposed hybrid threat model on controllers under our study. To perform this demonstration, we will exploit DoS attack on our network shown by our custom topology in Figure 6.1 and evaluate the impact of it on the performance metrics bandwidth and latency of Ryu and POX controllers. To perform this analysis, we use two types of traffic such as TCP and UDP traffic generated by using iperf and hping3 tools. For the use of comparison, we have started with the normal TCP and UDP traffic test and then continue with the attack traffic. Finally, we will measure the impact of DoS attack on the performance of the controllers by using bandwidth and network latency performance parameters.

6.3.1 Normal Traffic Generation and Testing

✓ **TCP Traffic Generation and Measurement**

As described in Section 6.2.2, the tools used for experimental simulation are ping, iperf and hping3 tools. In order to produce the TCP normal and attack traffics iperf and hping3 tools are used respectively in both POX and Ryu controllers. To make the test environment consistent, all the tools and network setups are similar for both controllers. In both controllers there are no default flow table entries, except they detected when their PACKET_IN event is triggered when they started up. We have selected two hosts randomly from the hosts in our network topology for experimentation as a server and a client. Throughout our test, h1 is selected as a client and h6 is selected as a server. In our network topology, the IP addresses for the client and the server hosts are configured as 10.0.0.1 and 10.0.0.6 respectively and server port to listen the requests is set as 5566.

- **Ryu controller**

As mentioned in Section 6.2.2, our controller is configured with the port number 6653 and IP address 127.0.0.1. The first tool we used to check the connectivity of hosts and to calculate the

response time of the server is ping. With this tool, we have transmitted 100 ICMP packets from client to server and then the Round Trip Time (RTT) is measured. RTT is a measurement that indicates the time it takes for a packet to travel from the source to the destination and back again. It is commonly used as a metric to evaluate network performance, particularly in terms of latency or delay. Here the RTT result reflects the time it takes for a network packet to make a round trip between the sender and receiver hosts. A low RTT indicates a network with good performance and low latency. It suggests that packets are traversing the network quickly, resulting in faster communication and responsiveness. On the other hand, a high RTT value indicates longer delays and slower network performance. When monitoring network performance, it is important to measure RTT values to get valuable information about the network performance. In this experiment, we have used the command h1- *ping –c 100 10.0.0.6* for transmitting 100 ICMP packets. For these packets, we have founded calculated minimum, average and maximum RTT values in Ryu controller are 0.041ms, 0.277ms, and 14.677ms as shown in Figure 6.5.

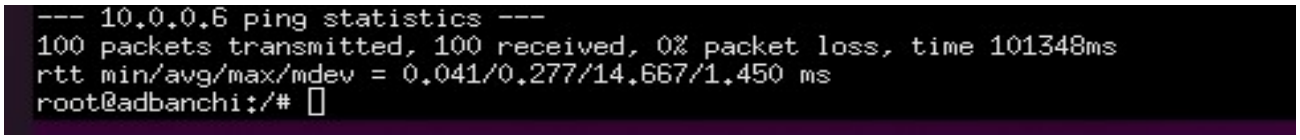


Figure 6.5 Ping summary of h1->h6 for Ryu controller

Other performance parameters, we have measured in this experiment are network bandwidth and delay or network latency. Bandwidth measurement is an important aspect of performance analysis in network systems. It refers to the measurement of the capacity or throughput of a network connection, indicating the amount of data that can be transmitted over the network in a given time period. Bandwidth is typically measured in bits per second (bps) and represents the maximum data rate that can be achieved on a network link. It determines the speed at which data can be transmitted between hosts across our SDN controller in our network. A high bandwidth measurement indicates a network connection with a larger capacity, capable of transmitting a greater amount of data in a shorter period. This implies faster data transfer rates and better overall network performance. On the other hand, a low bandwidth measurement suggests a network connection with limited capacity. This can lead to slower data transfer rates, increased latency, and potential congestion issues. By measuring bandwidth, we can determine the DoS attack impacts on the SDN controllers' performance. On the other hand, latency is another crucial factor in performance analysis in network systems. It refers to the time delay between the initiation of a data transfer and the moment the transfer is completed, and the time taken for a response to travel from the source to the

destination. It is measured in milliseconds (ms) and represents the time it takes for a data packet to travel from the sender to the receiver. In this performance analysis, latency measurement is used to assess the responsiveness and efficiency of a network connection through the SDN controllers under this study. It indicates the delay experienced in transmitting data across the SDN controller. A low latency measurement indicates a network connection with minimal delays. It suggests that data packets are traversing the network quickly, resulting in faster communication and responsiveness.

Conversely, a high latency measurement indicates longer delays in data transmission. This can result in noticeable delays, increased response times. By measuring latency, we can identify potential performance issues in relation to the exploited DoS attack on the SDN controlled by the controllers under our study. Both these parameters can be varied due to different internal and external factors, so in order to minimize the variations due to these factors, we have done some statistical calculations based on mean and standard deviation metrics. In TCP traffic analysis, we will analyze the bandwidth of our controllers. For doing bandwidth measurement, we have used iperf TCP traffic generation tool by using the command – "*iperf –s –p 5566 –i 1*" on the server host where flags '–s' indicates server mode, '–p' indicates port number for client server connection, '-i' indicates the time interval between periodic reports in msec and "*iperf –c 10.0.0.6 –p 5566 –t 15*" on the client host, where '–c' indicates client mode, '-p' indicates server port number, '-t' indicates duration of test in seconds. The execution of the commands and the resulting bandwidth measurement output screen shoot will be documented in Annex C– Experimental Results: TCP Normal Traffic Generation and Bandwidth Measurement for Ryu and POX controllers.

The delay parameter has been measured by using UDP network traffic latency measurement. Here with TCP traffic, we only have taken the bandwidth measurement. Bandwidth measurement of Ryu controller within 15 secs time duration and the resulting mean value of normal TCP traffic bandwidth for this controller is: 19.61Gbps and standard deviation is 1.8048Gbps. Figure 6.6 presents Wireshark capture result of normal TCP traffic flow through Ryu controller between two hosts.

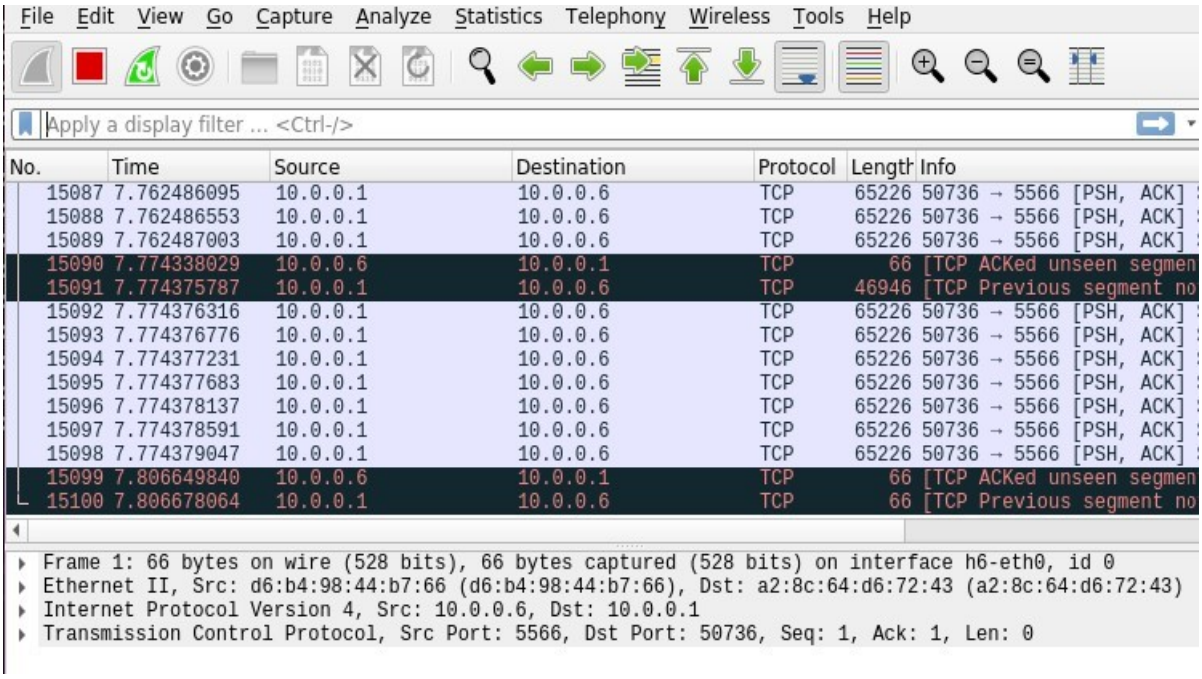


Figure 6 6 Normal TCP traffic flow through Ryu controller between client and server hosts

- **POX controller**

Similar steps, configurations and commands and experimental measurements for testing Ryu controller are followed here and resulting minimum, average and maximum RTT outputs from ping command for 100 ICMP packets between two hosts for POX controller are 0.035, 0.389 and 10.881 respectively as shown in Figure 6.7.

```
--- 10.0.0.6 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101251ms
rtt min/avg/max/mdev = 0.035/0.389/10.881/1.460 ms
```

Figure 6.7 Ping summary of h1->h6 for POX controller

Bandwidth measurement of this controller is also taken within 15 seconds time duration and its mean and standard deviation values are 16.67Gbps and the standard deviation is 3.3574 Gbps. Figure 6.8 shows Wireshark capture result of normal TCP traffic flow through POX controller between two hosts.

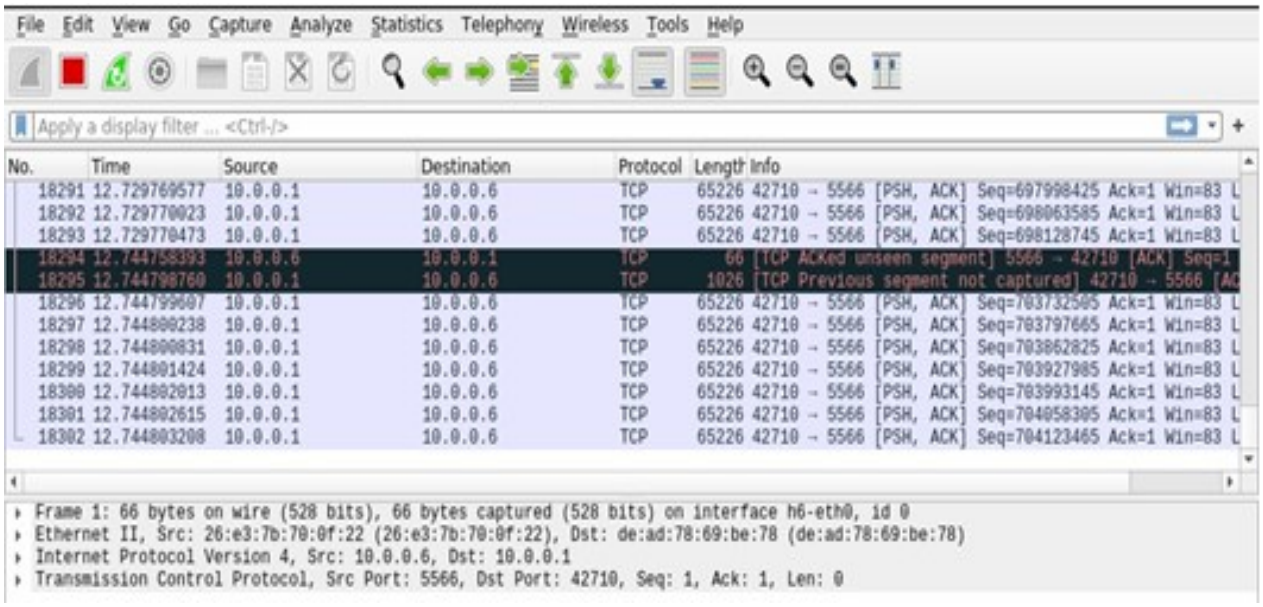


Figure 6.8 Normal TCP traffic flow through POX controller between client and server hosts

✓ **UDP Traffic Generation and Measurement**

In similar way with TCP traffic generation and measurement, UDP normal traffic is generated and measured. For generating normal UDP traffic, we have used iperf tool with the following command *–' iperf –s –u –p 5566 –i 1'* on the server host, where '-u' flag commands the generated traffic should be UDP traffic. Similarly, on the client host we have used the command- *'iperf –c 10.0.0.6 –u –p 5566 –t 15'.* The execution of this iperf command and the resulting output has been documented in Annex C– Experimental Results: UDP Normal Traffic Generation and Bandwidth Measurement for Ryu and POX controllers.

- **Ryu Controller**

As we have mentioned before, the delay parameter is measured by using normal UDP traffic flow from which we have taken the latency or jitter parameter measured form our network through iperf tool. From this measurement, we have gotten the result as 0.0276ms mean jitter value and 0.0200ms standard deviation value. Figure 6.9 indicates the normal UDP traffic flow through Ryu controller.

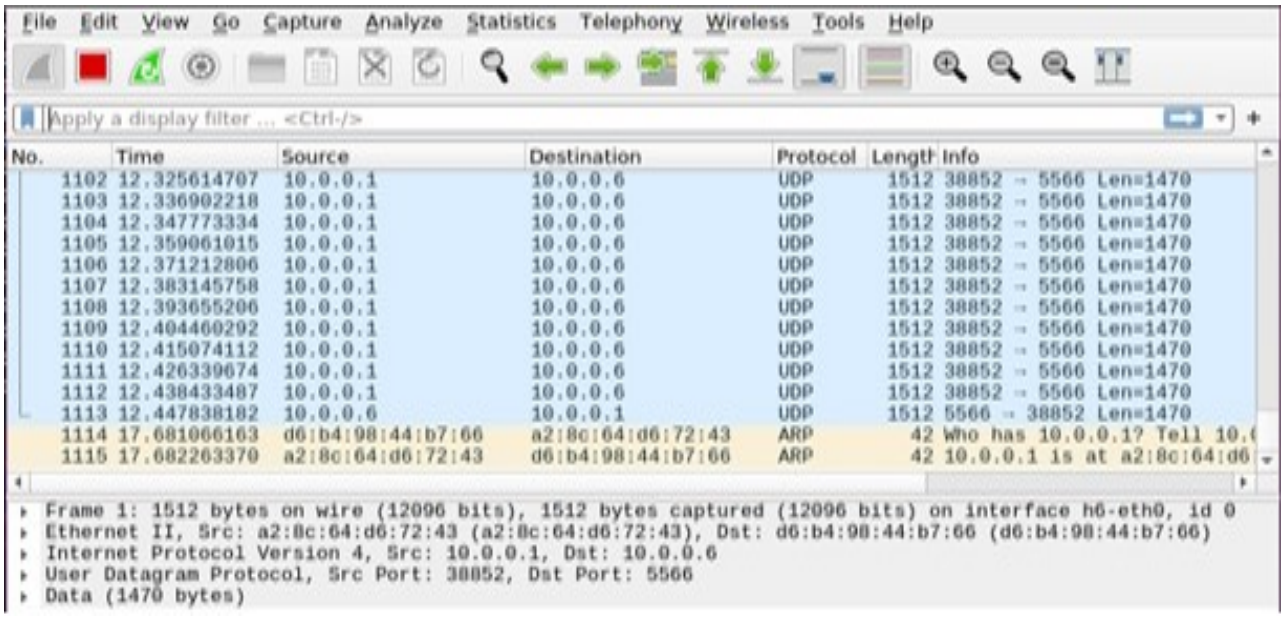


Figure 6.9 Normal UDP traffic flow through Ryu controller between client and server hosts

POX Controller

Similarly, normal UDP traffic generation and measurement of jitter parameter were conducted to know the comparative delay impact of the DoS attack on this controller by using iperf tool and the results for mean and standard deviation values of jitter for POX controller are: 0.0206ms and 0.0235ms respectively. Wireshark capturing result of UDP traffic flow through this controller is shown below in Figure 6.10.

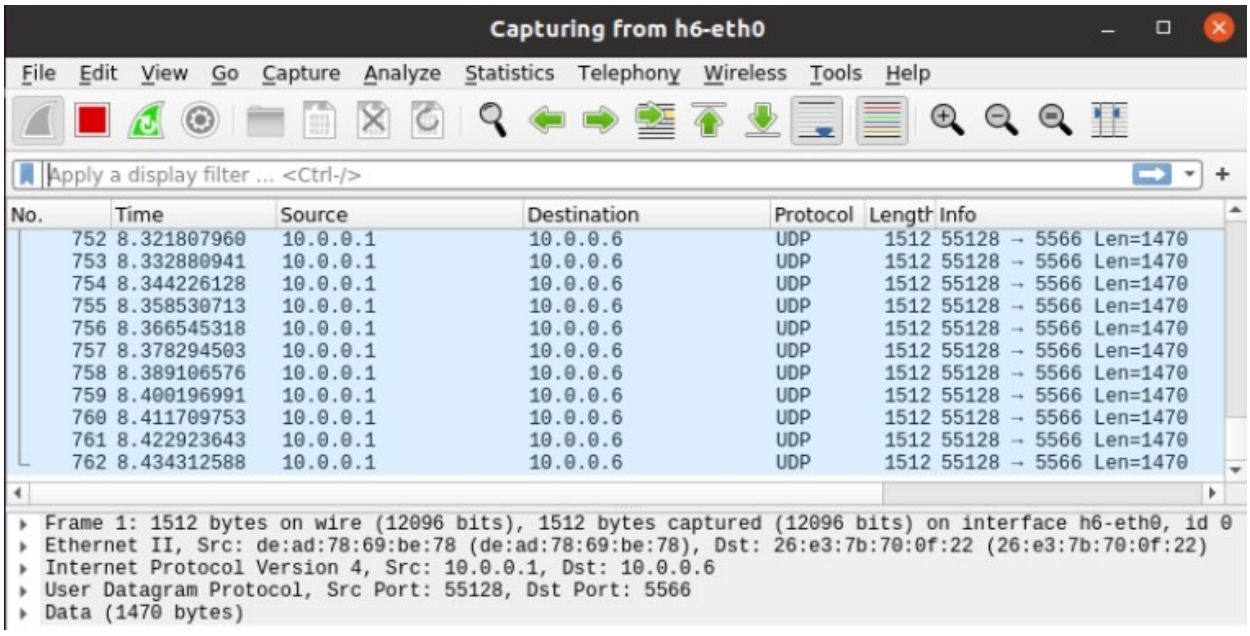


Figure 6.10 Normal UDP traffic flow through POX controller between client and server hosts

6.3.2 Attack Traffic Generation and Testing

As described earlier, the only executed attack for testing and evaluating our proposed hybrid threat model is DoS attack. To perform this attack on the network controlled by the controllers under our study and defined by our topology, we use hping3 attack or flood traffic generator tool to generate the attack traffic to measure the bandwidth and delay parameters we have used iperf tool ran on the server host. Here we assume that the client in our case host 1 which is identified by 10.0.0.1 IP address is a malicious user to generate and flood the controllers by using large amount of traffic flow and exploit DoS attack on the controllers and to hide itself the attacker uses random source address in the attack traffic flow. By using this assumption, we have executed the DoS attack using hping3 tool from the client xterm window and to measure the resulting output by using iperf tool from the server host. The command used to generate TCP and UDP attack traffics by using the following commands on the client xterm window.

'hping3 –c 10000 –p 5566 –flood –rand-source 10.0.0.6' for TCP flood traffic and *'hping3 --flood –a --rand-source –p 5566 10.0.0.6'* for UDP flood traffic. For measuring the results we have used the iperf command with server mode as shown on the normal TCP and UDP traffic flow generation and measurement. The resulting output for this attack traffic generation and measurement for both controllers are documented in Annex C– Experimental Results: TCP Attack Traffic Generation and Bandwidth Measurement for Ryu and POX controllers. And Annex C– Experimental Results: UDP Attack Traffic Generation and Jitter Measurement for Ryu and POX controllers. Now we will discuss the resulting measurements in each controllers as follows

✓ **TCP Traffic Generation and Measurement**

Here we will present TCP attack traffic generation and measurement in each controller.

- **Ryu Controller**

In Ryu controller, the settings in the controller and our network topology are all the same as the normal traffic generation and measurement for Ryu controller and by considering the above assumptions we have generated attack traffic from client with random identification and conduct the attack from client to the server identified by 10.0.0.6, then we have measured the bandwidth performance of the controller. The resulting output indicates that the mean value of bandwidth is 18.87Gbps and the standard deviation value of bandwidth is 1.7452Gbps. Figure 6.11 shows the Wireshark capture of TCP attack traffic. As shown from this figure, the attacker hides its identity (source address) and sends large number of packets to the victim host.

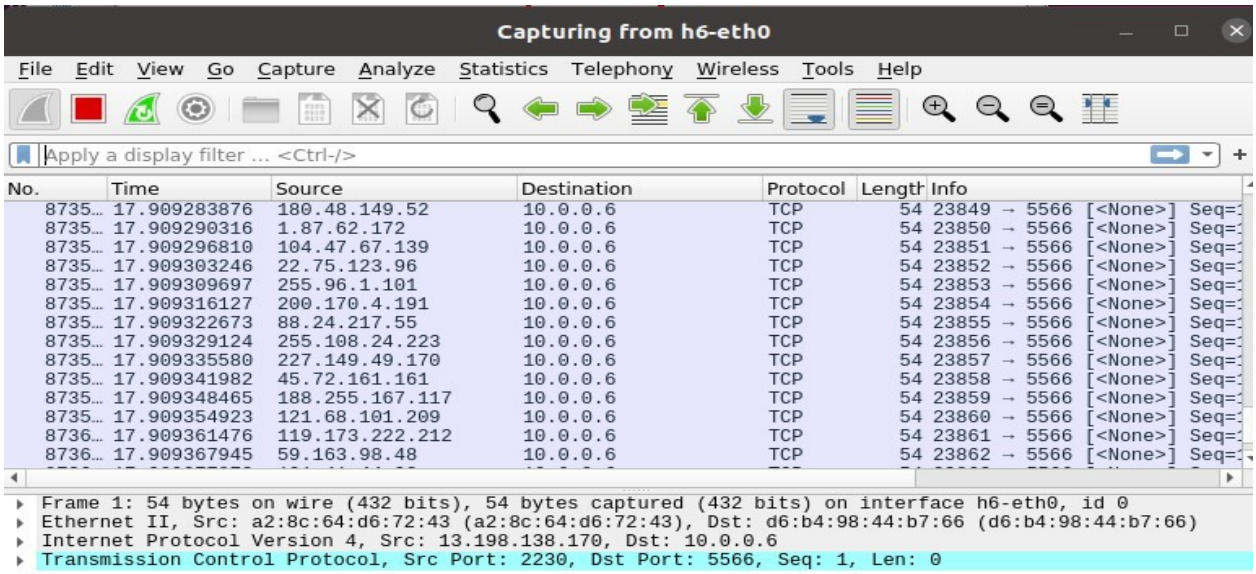


Figure 6.11 Attack TCP traffic flow through Ryu controller between client and server hosts

- **POX Controller**

Similarly in POX controller, all the settings in the controller and our network topology are the same as the normal traffic generation and measurement for it. And the attack traffic generation and measurement process in this controller is also similar to the Ryu controller. But the resulting output of POX controller under the executed DoS attack is different and that indicates mean value of bandwidth is 9.69Gbps and the standard deviation value of bandwidth is 2.2264Gbps. The Wireshark traffic capture of this controller which is shown at the right side of the Figure 6.12 and the number of packet transferred per second which flooded the network is shown in the left with I/O graph of the attack traffic data capture.

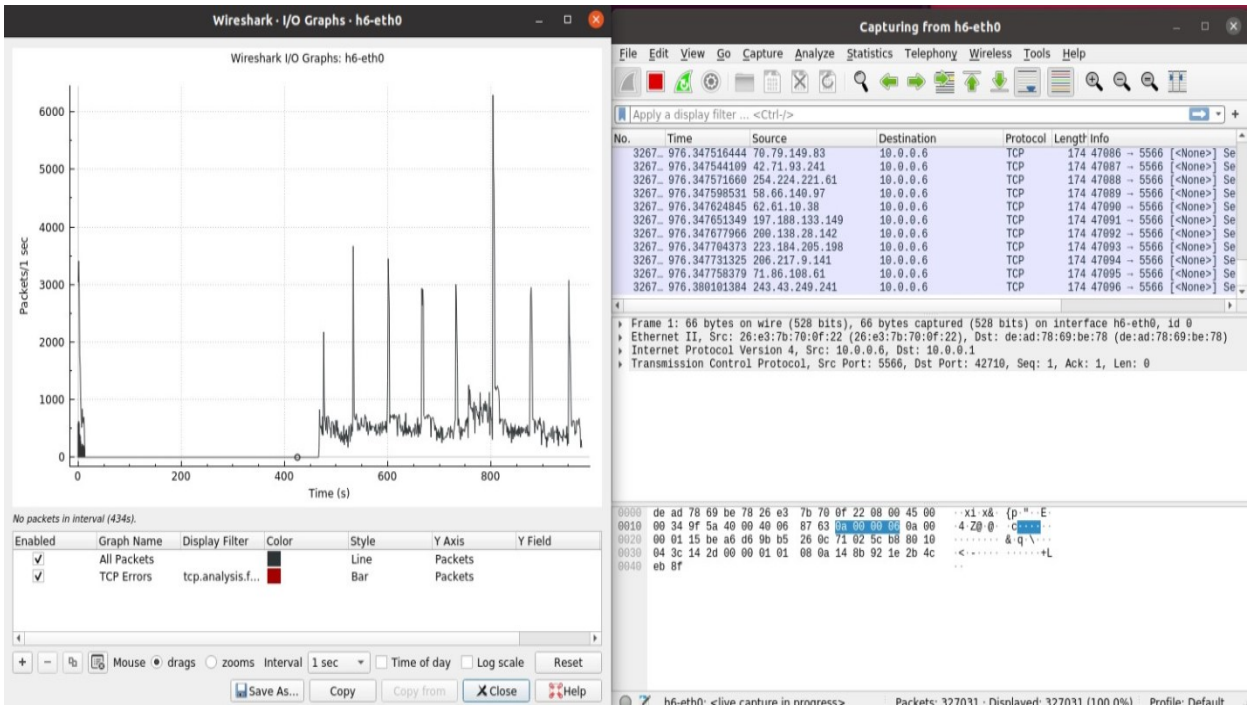


Figure 6.12 Attack TCP traffic flow through POX controller with its I/O graph

✓ **UDP Traffic Generation and Measurement**

As TCP attack traffic generation and measurement, we have followed similar procedure, use similar tool except adding '–a' flag which indicates the hping3 the required traffic is UDP as an attack traffic, the full command is documented above on the attack traffic generation and testing section of this document. From our measurement of UDP attack traffic, we obtained the mean value and standard deviation value of jitter for Ryu controller is 0.0292ms and 0.0213ms respectively and for POX controller the mean value and standard deviation value of jitter are 0.0246ms and 0.0265ms respectively.

6.4 Experimental Result Summary and Comparison

From our experiment, we have obtained the test results for RTT, bandwidth and jitter measured values of the controllers and we have summarized below in tabular form. For the RTT measurement of our experiment the result obtained for the two controllers is summarized in the table 6.1. In this result, RTT measurement is taken by transmitting 100 ICMP packets from h1 to h6 through Ryu and POX controllers, the obtained values for this parameter indicate that the time duration between sending a packet from h1 and receiving acknowledgement from h6. In this measurement, long time duration is interpreted as delayed transmission and short time duration is interpreted as fast transmission. Based on this interpretation, we can compare the controllers with their average RTT

values; the result shows that POX controller exhibits larger RTT value than Ryu controller. This indicates that packet transmission is delayed when POX controller is used. So, Ryu controller has better performance before DoS attack has been launched on these controllers.

Table 6.1 RTT value summary of ping test result in milliseconds

Controller	Minimum	Average	Maximum
Ryu	0.041	0.277	14.667
POX	0.035	0.389	10.881

Based on our research objective, we have tested and evaluated the analysis result of our proposed hybrid threat modeling method by using bandwidth and delay (jitter) performance parameters by exploiting one of the vulnerabilities that results from our proposed hybrid threat model, for doing so according to our analysis result both controllers are vulnerable to DoS attack and based on the selection criteria mentioned in section 6.1 of this Chapter, the controllers under this study are tested against DoS attack. The impact of the attack on the performance parameters bandwidth and jitter in relative to normal traffic are summarized in Table 6.2 and 6.3.

Table 6.2 TCP Traffic bandwidth comparison between normal and attack traffic for Ryu and POX

Controller	Traffic	Bandwidth Mean (Gbps)	Bandwidth Standard Deviation (Gbps)
Ryu	Normal Traffic	19.61	1.8048
	Attack Traffic	18.87	1.7452
POX	Normal Traffic	16.67	3.3574
	Attack Traffic	9.69	2.2264

From TCP bandwidth result, in Ryu controller mean bandwidth variation between client and server hosts with attack traffic is approximately 0.74Gbps lower than from normal traffic bandwidth and it is around 6.98Gbps in case of POX controller which indicates significantly higher drop of bandwidth. This proves that DoS attack significantly degraded the performance of POX than Ryu controller. From our result DoS attack is possible on both controllers but the impact is severe for POX controller than Ryu controller in terms of bandwidth. Figure 6.13 show that bandwidth slightly degraded between hosts whenever flooding attack increases towards the hosts in Ryu controller.

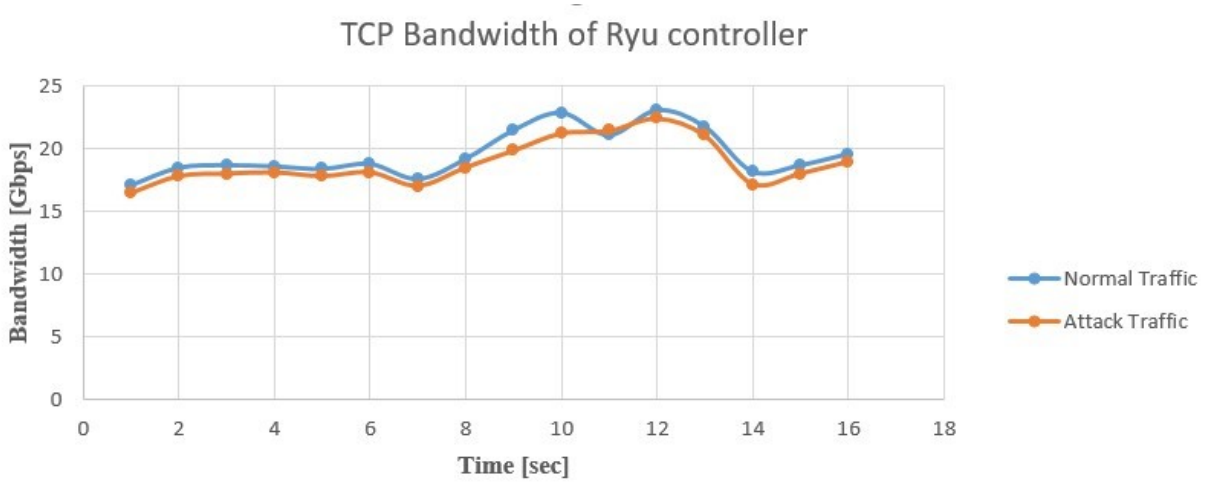


Figure 6.13 TCP Normal Vs Attack Traffic bandwidth of Ryu Controller

Moreover, Figure 6.14 shows clearly that TCP bandwidth between hosts significantly degraded whenever flooding attack increases towards these hosts in POX controller.

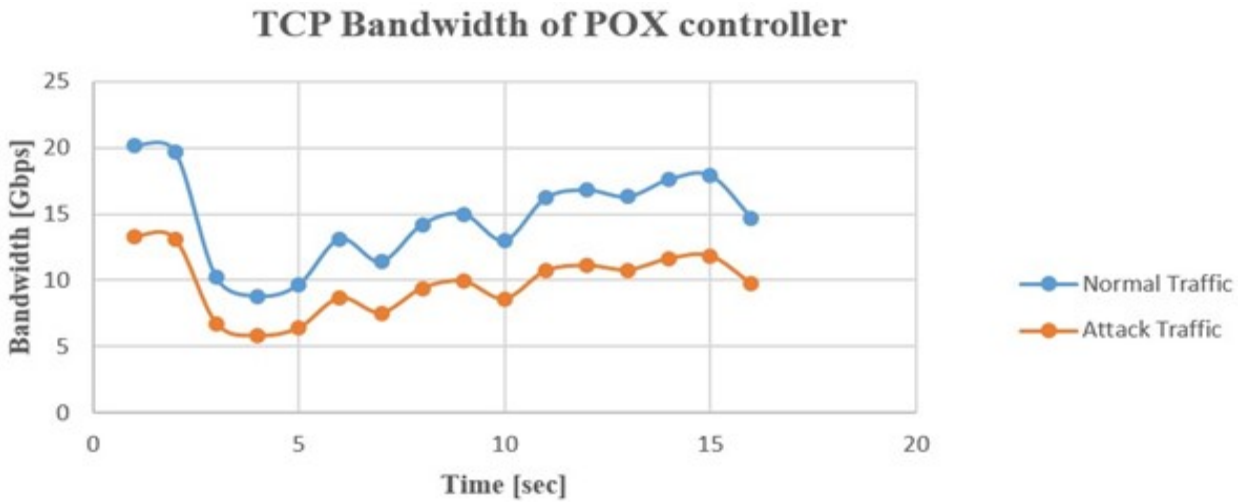


Figure 6.14 TCP Normal Vs Attack traffic bandwidth of POX controller

Table 6.3 UDP Traffic jitter comparison between normal and attack traffic for Ryu and POX

Controller	Traffic	Jitter mean (ms)	Jitter Standard Deviation (ms)
Ryu	Normal Traffic	0.0276	0.0200
	Attack Traffic	0.0292	0.0213
POX	Normal Traffic	0.0206	0.0235
	Attack Traffic	0.0246	0.0265

Like TCP, UDP traffic also generated and examined for the impact of DoS attack on the performance analysis parameter jitter in order to measure traffic delay in Ryu and POX controllers. Based on the result from our experimentation, in Table 6.3, we summarize the jitter measurement for normal and attack UDP traffic in Ryu and POX controllers. In Ryu controller the mean jitter measurement difference between normal and attack UDP traffic is 0.0016ms, this figure is almost negligible difference on jitter which indicates the resistance of Ryu controller against DoS attack towards traffic delay caused by the attack. Similarly, POX exhibit a very small mean value difference between normal and attack traffic on jitter parameter i.e. 0.004ms, it also indicates that the DoS attack has not significantly affect POX controller towards traffic delay. The delay effects on both controllers are shown in Figure 6.14 and Figure 6.15. As shown in Figure 6.15 Ryu exhibits pick delay value around 0.081ms at time 8secs.

Figure 6.15 UDP Normal vs Attack traffic delay of Ryu controller

Similarly, POX exhibits pick delay value around 0.109ms at time of 5 seconds and it also exhibit big value difference between delays for normal and attack traffic around 0.033ms at 8 seconds as shown in Figure 6.16.

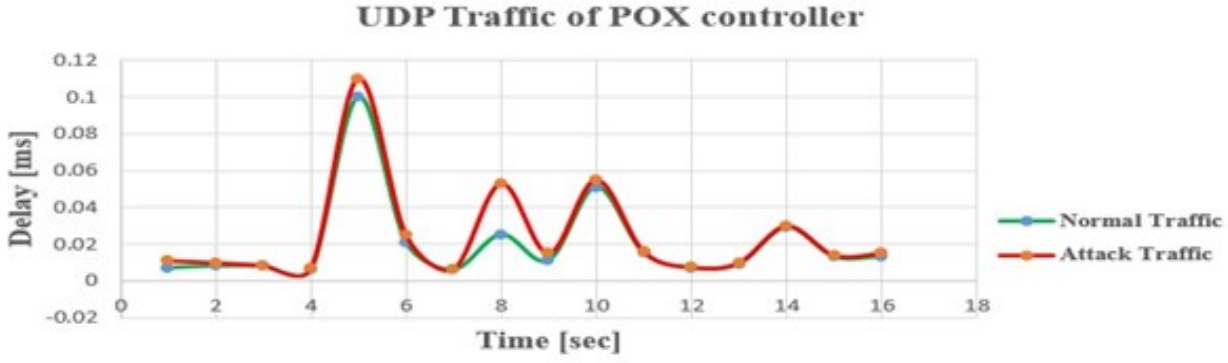


Figure 6.16 UDP Normal Vs Attack Traffic of POX controller

6.5 Summary

Based on the analysis result of our proposed hybrid threat model, we have experimentally tested the effectiveness of our proposed hybrid threat model on one of the vulnerability result DoS which was selected by the criteria mentioned in Section 6.1. To evaluate the result, the impact of DoS attack on the performance of Ryu and POX controllers is studied experimentally by using performance parameters bandwidth and delay (jitter). As we have shown from our experimental results, both Ryu and POX controllers are vulnerable to DoS attacks. The DoS attack had a more severe impact on the performance of the POX controller compared to the Ryu controller. The drop in bandwidth for the POX controller was significantly higher than for the Ryu controller, indicating that the performance degradation was more pronounced. The impact of the DoS attack on traffic delay (jitter) was relatively small for both controllers. The mean jitter measurement differences between normal and attack traffic were very small for both Ryu and POX controllers, indicating that the attack did not significantly affect traffic delay. The experimental results validate the effectiveness of our proposed hybrid threat model for analyzing the security of SDN controllers. The evaluation of the DoS attack vulnerability demonstrates that the ability of the model to identify and assess potential threats with better threat coverage and granularity when compared to the STRIDE only method in previous studies.

Chapter 7: Conclusion, Recommendation and Future Works

7.1 Conclusion

In conclusion, the aim of this thesis work was to overcome the limitations of the STRIDE only threat model when analyzing the security of software defined network (SDN) controllers. The problem we addressed was lack of granularity, limited threat coverage, and moderately high false negative rate associated with the STRIDE only approach. To improve the effectiveness of the STRIDE only model used in previous studies to analyze the security of SDN controllers, our research focused on designing a hybrid threat model that combines both STRIDE and attack tree models. This hybrid model provides a more comprehensive and detailed analysis of the security threats faced by SDN controllers by leveraging the strengths of both models. The STRIDE model offers a systematic approach for identifying and categorizing security threats based on six dimensions, while attack trees provide a hierarchical representation of attack scenarios, allowing for a more detailed analysis of vulnerabilities and attack paths. By combining these models, the hybrid threat model provides a better rounded understanding of the security issues in SDN controllers. It enables the identification of a broader range of threats, including those that may have been overlooked by the STRIDE only model. Additionally, our proposed model offers a more detailed analysis of attack scenarios, facilitating a better understanding of the potential impact and severity of each threat. We have assessed security vulnerabilities in Ryu and POX SDN controllers by using our proposed hybrid model. Our analysis revealed several vulnerabilities, including tampering and information disclosure attacks on the data flows of the Ryu controller, and spoofing, tampering, denial of service, and elevation of privilege attacks on its core. The POX controller core was also found to be vulnerable to information disclosure, repudiation, and denial of service attack types. To evaluate the effectiveness of our proposed hybrid threat model in security analysis of controllers compared to the STRIDE only threat model, we conducted an evaluation by considering two scenarios based on our findings as described in Section 5.2.3. To further evaluate our proposed hybrid threat model, we conducted an experiment to simulate the impact of a denial of service (DoS) attack on the performance parameters of bandwidth and delay in the controllers. We selected the DoS attack as it was common security vulnerability in both controllers and could not be detected by the STRIDE only method in previous studies. In the experiment, we used TCP and UDP normal and attack traffic flows generated by tools such as iperf and hping3. The results showed significant bandwidth degradation in the POX controller but only a small delay effect in

both controllers. This experiment also demonstrated that our proposed model can detect vulnerabilities in SDN controllers better than the STRIDE only model.

7.2 Recommendation

In organizations which implement SDN, regular review and update of security measures is essential to adapt emerging threats and vulnerabilities. Conducting regular vulnerability assessments on the SDN controller infrastructure is always essential to identify any new vulnerabilities or misconfigurations and allow for timely remediation. Based on our analysis result, to mitigate the vulnerabilities found from our analysis we recommend implementation of secure configurations for the SDN controllers, including strong authentication mechanisms and proper access controls. Use of strong and unique passwords for administrative accounts and considering multifactor authentication implementation for added security and encryption mechanisms, such as Transport Layer Security (TLS), to protect sensitive data transmitted between the SDN controller and other network components. This helps to prevent eavesdropping and information disclosure attacks. Enabling comprehensive logging and monitoring of the SDN controller and associated network components is also mitigate the resulting vulnerabilities. In addition, centralized log collection and analysis are also recommended to identify any suspicious activities or potential security breaches and implementation of real time alerting to respond promptly to any security incidents in SDN. Finally, deployment and use of intrusion detection and prevention systems (IDS/IPS) to monitor network traffic and to detect any malicious activities or anomalies are recommended. IDS/IPS systems can help organizations to identify and block potential threats or attacks and provide an additional layer of defense.

7.3 Future Works

To evaluate our proposed hybrid threat model, we have made the security analysis and also experimental test on only Ryu and POX controllers. In the future, we will evaluate and test our model on other SDN controllers. Moreover, there exist vulnerability scoring models, we will integrate our proposed model with one of the vulnerability scoring models to incorporate quantitative feature in our model in addition to its qualitative nature to assess the security of SDN controllers.

References

[1] S. H. Haji et al., "Comparison of Software Defined Networking with Traditional Networking"," Asian Journal of Research in Computer Science, vol. 9, no. 2, pp. 1–18, 2021.

[2] N. Mckeown et al., "Openflow: Enabling Innovation in Campus Networks," Acm Sigcomm Computer ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, 2008.

[3] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software defined networking," IEEE communications surveys and tutorials, vol. 16, no. 4, pp. 69–74, 2014.

[4] Z. Shu, J. Wan, D. Li, J. Lin, A. V. Vasilakos, and M. Imran, "Security in software-defined networking: Threats and countermeasures," Mob. Netw. Appl., vol. 21, no. 5, pp. 764–776, 2016.

[5] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in IEEE Local Computer Network Conference, 2010.

[6] Karan, Narayan, and P. S. Hiremath, "Detection of DDoS attacks in software defined networks," in 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), 2018.

[7] R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of Microsoft's threat modeling technique," Requir. Eng., vol. 20, no. 2, pp. 163–180, 2015.

[8] "A hybrid threat modeling method," SEI Digital Library. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=516617. [Accessed: 30-Oct-2023].

[9] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, "STRIDE-based threat modeling for cyber-physical systems," in 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2017.

[10] A. Karahasanovic, P. Kleberger, and M. Almgren, "Adapting threat modeling methods for the automotive industry," in escar Europe conference | Embedded Security in Cars, 2017.

[11] Z. Shi, K. Graffi, D. Starobinski, and N. Matyunin, "Threat modeling tools: A taxonomy," IEEE Secur. Priv., vol. 20, no. 4, pp. 29–39, 2022.

[12] B. Potteiger, G. Martins, and X. Koutsoukos, "Software and attack centric integrated threat modeling for quantitative risk assessment," in Proceedings of the Symposium and Bootcamp on the Science of Security, 2016.

[13] B. Schneier, “Schneier on security,” Schneier.com. [Online]. Available: https://www.schneier.com/academic/archives/1999/12/attack_trees.html. [Accessed: 30-Oct-2023].

[14] D. Beaulaton, N. B. Said, I. Cristescu, and S. Sadou, “Security analysis of IoT systems using attack trees,” in Graphical Models for Security, Cham: Springer International Publishing, 2019, pp. 68–94.

[15] T. Ucedavélez and M. M. Morana, Risk centric threat modeling: Process for Attack Simulation and Threat Analysis. Wiley Publishing.

[16] A. A. Singh and K. S. Singh, “Network Threat Ratings in Conventional DREAD Model Using Fuzzy Logic,” International Journal of Computer Science Issues, vol. 9, no. 3, 2012.

[17] W. V. Systems, S. Prakash, K. A. Söderberg-Rivkin, P. Kadhirvelan, S. Söderberg-Rivkin, and T. Olovsson, “Threat modelling and risk assessment,” Chalmers.se. [Online]. Available: https://publications.lib.chalmers.se/records/fulltext/202917/202917.pdf. [Accessed: 30-Oct-2023].

[18] D. P. Gilliam and J. D. Powell, “Integrating a flexible modeling framework (FMF) with the network security assessment instrument to reduce software security risk,” in Proceedings eleventh IEEE international workshops on enabling technologies, pp. 153–158.

[19] K. Srinivasan, S. Matheswaran, and P. Sengodan, “Security threats and countermeasures in software defined networking,” JETIR, vol. 6, 2019.

[20] T. Wan, A. Abdou, and P. C. Oorschot, “A framework and comparative analysis of control plane security of SDN and Conventional networks,” IEEE COMST, 2018.

[21] A. Konev, A. Shelupanov, M. Kataev, V. Ageeva, and A. Nabieva, “A survey on threat-modeling techniques: Protected objects and classification of threats,” Symmetry (Basel), vol. 14, no. 3, p. 549, 2022.

[22] N. Shevchenko, B. R. Frye, C. Woody, and Carnegie Mellon University Software Engineering Institute, “Threat Modeling: Evaluation and recommendations,” 2018.

[23] F. Shull and N. R. Mead, “Cyber threat modeling: An evaluation of three methods,” SEI Blog. [Online]. Available: http://insights.sei.cmu.edu/blog/cyber-threat-modeling-an-evaluation-of-three-methods/. [Accessed: 30-Oct-2023].

[24] A. L. Opdahl and G. Sindre, “Experimental comparison of attack trees and misuse cases for security threat identification,” Inf. Softw. Technol., vol. 51, no. 5, pp. 916–932, 2009.

[25] Q. Ilyas and R. Khondoker, “Security analysis of FloodLight, ZeroSDN, beacon and POX SDN controllers,” in SDN and NFV Security, Cham: Springer International Publishing, 2018, pp. 85–98.

[26] T. Hu, P. Yi, and Y. Hu, “An Analysis ofSDNControllers Software,” Wiley 5G Ref. Wiley, pp. 1–19, 29-Dec-2019.

[27] H. Omotunde, R. Ibrahim, a. R. Of, and I. Modelling, “Ibrahim “A REVIEW OF THREAT MODELLING and ITS HYBRID APPROACHES TO SOFTWARE SECURITY TESTING,” ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 23, 2015.

[28] “How to use *MiniEdit*, Mininet’s graphical user interface,” https://www.brianlinkletter.com [Online]. Available: https://www.brianlinkletter.com/2015/04/how-to-use-miniedit-mininets-graphical-user-interface\.[Accessed: 20-Sep-2023].

[29] A. Tirumala, “Iperf: The TCP/UDP bandwidth measurement tool,” http://dast. nlanr. net/Projects/Iperf/, 1999.

[30] D. Adams, “DOS flood with hping3,” Linuxhint.com. [Online]. Available: https://linuxhint.com/hping3/. [Accessed: 30-Oct-2023].

[31] R. Shimonski, The Wireshark field guide: analyzing and troubleshooting network traffic. Newnes, 2013.

[32] R. Khondoker, Feature-based comparison and selection of Software Defined Networking (SDN) controllers.”, World Congress on Computer applications 6 security analysis of FloodLight, ZeroSDN, beacon and POX SDN Controllers 101 and Information Systems (WCCAIS). IEEE, 2014.

[33] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” in Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, 2013.

[34] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” IEEE Commun. Surv. Tutor., vol. 16, no. 3, pp. 1617–1634, 2014.

[35] S. Li et al., “Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability,” IEEE Netw., vol. 31, no. 2, pp. 58–66, 2017.

[36] S. Schaller and D. Hood, “Software defined networking architecture

standardization," Comput. Stand. Interfaces, vol. 54, pp. 197–202, 2017.

[37] M. Á. Pérez, N. Y. S. Barrera, E. R. Losada, and G. M. Sánchez, "State of the art in software defined networking (SDN)," Visión electrónica, vol. 13, no. 1, pp. 178–194, 2019.

[38] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, Software-defined networking (SDN): Layers and architecture terminology. No. rfc7426. 2015.

[39] M. Kuzniar, P. Peresini, and D. Kostic, "What you need to know about SDN control and data planes," No. REP_WORK, 2014.

[40] M. Paliwal, D. Shrimankar, and O. Tembhurne, "Controllers in SDN: A Review Report," IEEE Access, vol. 6, pp. 36256–36270, 2018.

[41] S. Ahmad and A. H. Mir, "SDN Interfaces: protocols, taxonomy and challenges," Int. J. Wirel. Microwave Technol.(IJWMT), vol. 12, no. 2, pp. 11–32, 2022.

[42] C. Röpke and T. Holz, "On network operating system security: ON NETWORK OPERATING SYSTEM SECURITY," Int. J. Netw. Manage., vol. 26, no. 1, pp. 6–24, 2016.

[43] T. Li, J. Chen, and H. Fu, "Application Scenarios based on SDN: An Overview," J. Phys. Conf. Ser., vol. 1187, no. 5, p. 052067, 2019.

[44] S. Badotra and J. Singh, "Creating firewall in transport layer and application layer using software defined networking," in Innovations in Computer Science and Engineering, Singapore: Springer Singapore, 2019, pp. 95–103.

[45] A. Lara and L. Quesada, "Performance Analysis of SDN Northbound Interfaces," in 2018 IEEE 10th Latin-American Conference on Communications (LATINCOM), 2018.

[46] L. Boukraa, S. Mahrach, K. El Makkaoui, and R. Esbai, "SDN southbound protocols: A comparative study," in Lecture Notes on Data Engineering and Communications Technologies, Cham: Springer International Publishing, 2023, pp. 407–418.

[47] N. Shahzad, G. Mujtaba, and M. Elahi, "Benefits, security and issues in software defined networking (SDN)," NUST Journal of Engineering Sciences, vol. 8, no. 1, pp. 38–43, 2015.

[48] V. Thirupathi, C. H. Sandeep, N. Kumar, and P. P. Kumar, "A comprehensive review on sdn architecture, applications and major benifits of SDN," International Journal of Advanced Science and Technology, vol. 28, no. 20, pp. 607–614, 2019.

[49] A. H. Shamsan and A. R. Faridi, "Security Issues and Challenges in SDN," in Communications in Computer and Information Science, Singapore: Springer Singapore,

2021, pp. 515–535.

[50] A. A. Semenovykh and O. R. Laponina, “Comparative analysis of SDN controllers,” International Journal of Open Information Technologies, vol. 6, no. 7, pp. 50–56, 2018.

[51] H. G. Ahmed and R. Ramalakshmi, “Performance analysis of centralized and distributed SDN controllers for load balancing application,” in 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018.

[52] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, “Distributed SDN controller system: A survey on design choice,” Comput. Netw., vol. 121, pp. 100–111, 2017.

[53] H. K. Ravuri, M. T. Vega, J. van der Hooft, T. Wauters, and F. De Turck, “A scalable hierarchically distributed architecture for next-generation applications,” J. Netw. Syst. Manag., vol. 30, no. 1, 2022.

[54] D. B. Hoang and M. Pham, “On software-defined networking and the design of SDN controllers,” in 2015 6th International Conference on the Network of the Future (NOF), 2015.

[55] J. Medved, R. Varga, A. Tkacik, and K. Gray, “OpenDaylight: Towards a Model-Driven SDN Controller architecture,” in Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, 2014.

[56] P. Berde et al., “ONOS: towards an open, distributed SDN OS,” in Proceedings of the third workshop on Hot topics in software defined networking, 2014, pp. 1–6.

[57] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, “Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers,” in 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks), 2016.

[58] S. Midha and K. Tripathi, “Assessing the Quality of Service of POX Controller in SDN,” International Journal of Computational Engineering Research (IJCER), vol. 8, no. 08, pp. 21–25, 2018.

[59] Y. Tseng, F. Naït-Abdesselam, and A. Khokhar, “A comprehensive 3-dimensional security analysis of a controller in software-defined networking,” Secur. Priv., vol. 1, no. 2, p. e21, 2018.

[60] T. M. Ghazal, M. A. M. Afifi, and D. Kalra, “Security vulnerabilities, attacks, threats and the proposed countermeasures for the Internet of Things applications,” Solid State Technology, vol. 63, no. 1s, 2020.

[61] L. O. Nweke and Stephen, “A review of asset-centric threat modelling approaches,” Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 2, 2020.

[62] G. Viswanathan and Prabhu, “A hybrid threat model for system-centric and attack-centric for effective security design in SDLC,” Web Intell., vol. 19, no. 1–2, pp. 1–11, 2021.

[63] N. R. Mead, F. Shull, K. Vemuru, and O. Villadsen, “A hybrid threat modeling method,” in Carnegie MellonUniversity-Software Engineering Institute, 2018.

[64] Q. Li and Y.-L. Chen, In Modeling and Analysis of Enterprise and Information Systems. Berlin, Heidelberg: Springer, 2009.

[65] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, Uncover Security Design Flaws Using The STRIDE Approach. 2006.

[66] W. Mbaka and K. Tuma, “A replication of a controlled experiment with two STRIDE variants,” arXiv [cs.CR], 2022.

[67] J. Luna, N. Suri, and I. Krontiris, “Privacy-by-design based on quantitative threat modeling,” in 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), 2012.

[68] T. Frankeline, “Attack modeling and risk assessments in software Defined networking (SDN),” 2019.

[69] R. Klöti, V. Kotronis, and P. Smith, “OpenFlow: A security analysis,” in 2013 21st IEEE International Conference on Network Protocols (ICNP), IEEE, 2013, pp. 1–6.

[70] H. Xu, J. Su, X. Zong, and L. Yan, “Attack identification for software-defined networking based on attack trees and extension innovation methods,” in 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), vol. 1, IEEE, 2017, pp. 485–489.

[71] F. Rui, “Modeling extenics innovation software by intelligent service components,” The Open Cybernetics & Systemics Journal, vol. 8, no. 1, 2014.

[72] T. V. Mehrdad and T. Phan, “Probability analysis of successful cyber attacks in sdn-based networks,” in 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2018, pp. 1–6.

[73] L. Yao, P. Dong, T. Zheng, H. Zhang, X. Du, and M. Guizani, “Network security analyzing and modeling based on Petri net and Attack tree for SDN,” in 2016 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2016, pp. 1–5.

[74] “AT Tools,” Drupal.org, 05-Dec-2013. [Online]. Available: https://www.drupal.org/project/at_tools. [Accessed: 30-Oct-2023].

[75] A. Yathuvaran, AT-AT: AT-AT (Attack Tree Analysis Tool) is a application that allows users to develop and analyze attack trees. The overall goal is to automatically generate a set of possible attack scenarios that can be used to provide guidance for how to improve the design of the system to which the attack tree belongs to. .

[76] S. Wierckx, “Examining attack tree tools, how do they compare?,” Toreon - Business driven cyber consulting, 19-Aug-2022. [Online]. Available: https://www.toreon.com/examining-attack-tree-tools/. [Accessed: 30-Oct-2023].

[77] “Setup TLS connection — Ryu 4.34 documentation,” Readthedocs.io. [Online]. Available: https://ryu.readthedocs.io/en/latest/tls.html. [Accessed: 30-Oct-2023].

[78] “Welcome to RYU the Network Operating System(NOS) — Ryu 4.34 documentation,” Readthedocs.io. [Online]. Available: https://ryu.readthedocs.io/en/. [Accessed: 30-Oct-2023].

[79] GitHub: Let’s build from here. .

[80] S. Shin et al., “Rosemary: A Robust, Secure, and Highperformance Network Operating System,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS ’14, New York, NY, USA: ACM, 2014, pp. 78–89.

[81] https://github.com/faucetsdn, RYU:invoking application and Configuration. .

[82] “IEEE 802.1x / RADIUS,” Elektronikkompendium.de. [Online]. Available: http://www.elektronikkompendium.de/sites/net/1409281.htm. [Accessed: 30-Oct-2023].

[83] D. Mattos and O. Duarte, “AuthFlow: authentication and access control mechanism for software defined networking,” Ann Telecommun, vol. 71, no. 11–12, pp. 607–615, 2016.

Annexes

Annex A - my_custom_topology.py script code

To create our custom topology, we have used python code, the following was written and runs on mininet and connect with controllers remotely. By this code we have created a single controller (remote), two openflow switches and 6 hosts with the link structure located on the graphical topology diagram of our topology by using miniedit in this document. Our custom topology code is created by using the following python script code and saved with the file name my_custom_topology.py and run as python3 my_custom_topology.py command.

```
#my_custom_topolgy code

from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch
from mininet.cli import CLI


def create_custom_topology():
    # Create a Mininet object
    net = Mininet(controller=Controller, switch=OVSSwitch)

    # Add a controller
    c0 = net.addController('c0')

    # Add two switches
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')

    # Add six hosts
    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')
    h4 = net.addHost('h4')
    h5 = net.addHost('h5')
    h6 = net.addHost('h6')

    # Add links between switches and hosts
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)
    net.addLink(s2, h4)
    net.addLink(s2, h5)
    net.addLink(s2, h6)

    # Start the network
    net.build()
    net.start()

    # Connect the switches to the controller
    s1.start([c0])
    s2.start([c0])
```

```
    # Open the Mininet command-line interface
    CLI(net)

    # Stop the network
    net.stop()


if __name__ == '__main__':
    create_custom_topology()
```

Annex B – Mininet, POX and Ryu SDN controllers' installation commands

Mininet, Ryu and .POX controllers' installation steps on Ubuntu 20.04 virtual machine OS.

✓ **Mininet emulator installation commands**

To install mininet from source, we have use the following installation commands

$sudo apt-get update

$sudo apt-get upgrade

$sudo apt-get install git

$git clone https://github.com/mininet/mininet

$mininet/util/ install.sh -a

After installation, to check whether mininet installed correctly or not by

$mn command

✓ **POX controller installation commands and Testing commands**

We have installed POX controller from source by using the following commands:

POX controller installation commands, we have used for our installation are

$sudo apt install –y python3.pip

$git clone https://github.com/noxrepo/pox

After installation we have tested whether POX installed successfully or not, we have opened 2 terminal windows and then use the following commands each on separate terminal

On first terminal, we have used

$python3 ./pox.py forwarding.l2_learning

On the second terminal, we have used

$sudo mn –controller remote

✓ **Ryu controller installation commands and Testing**

To install Ryu controller, we have used the following commands

$sudo apt install –y python3.pip

$git clone https:/github.com/osrg/ryu.git

$sudo python3 ./setup.py install

$sudo pip3 install –upgrade ryu

After installation we have tested whether ryu installed successfully or not, we have opened 2 terminal windows and then use the following commands each on separate terminal

On first terminal, we have used

$ryu-manager ryu/ryu/apps/simple_switch_13.py

On the second terminal, we have used

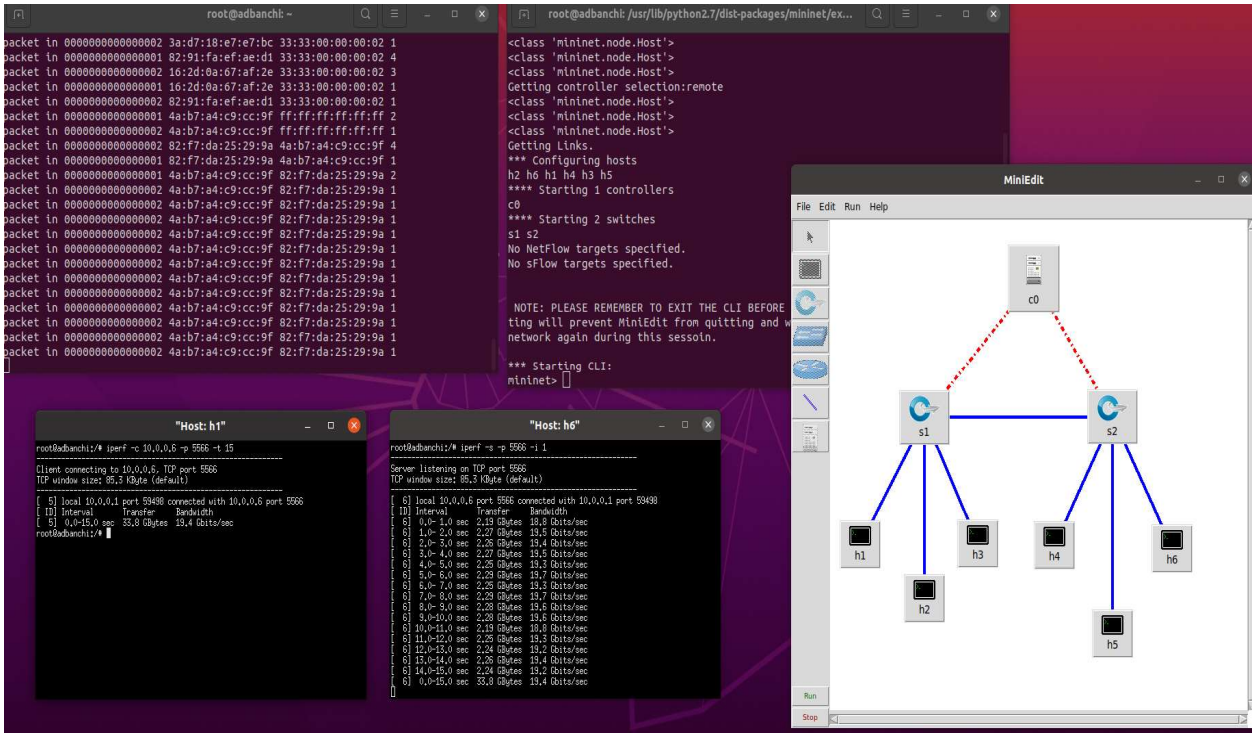
$sudo mn –controller remote

Then successful connection of ryu controller and the mininet default topology created by mn command by using ryu controller as remote controller successfully created.

Annex C– Experimental Results

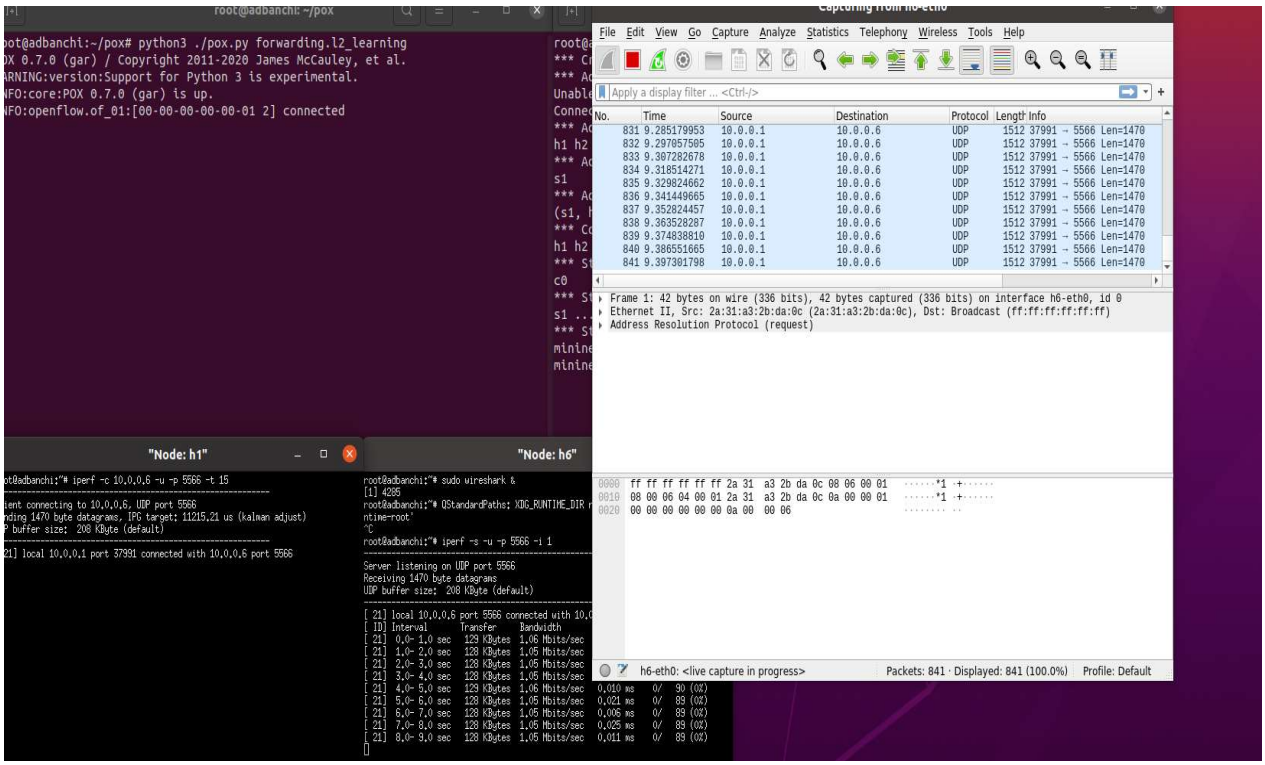
By using the commands and tools, we have described in detail on this thesis document in experimental testing and evaluation section. Here we will document a sample screenshot for each traffic generation and automatic bandwidth measurements in Annex Figures: Annex C.1, C.2, C.3, and C.4

TCP Normal Traffic Generation and Bandwidth Measurement for controllers



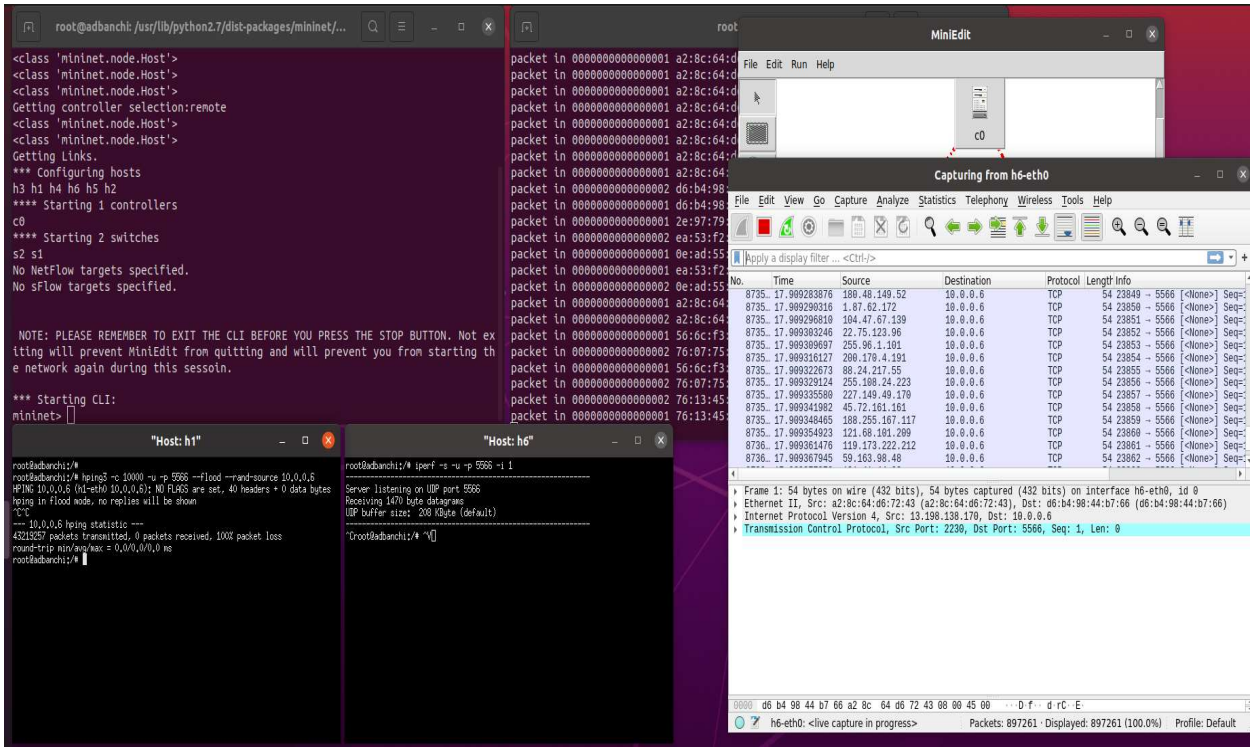
Annex C. 1: TCP Normal Traffic Generation and Bandwidth Measurement of controllers

UDP Normal Traffic Generation and Delay (jitter) Measurement for controllers



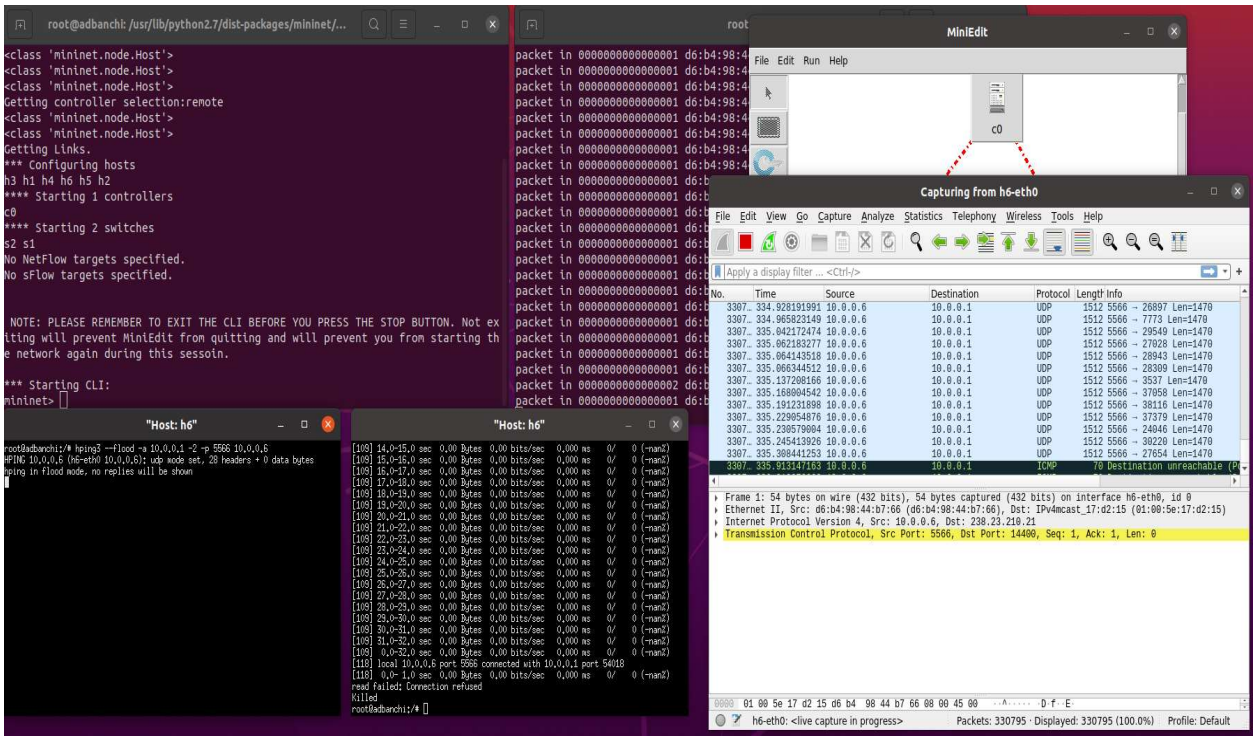
Annex C. 2: UDP Normal Traffic Generation and Jitter measurement of controllers

TCP Attack Traffic Generation and Bandwidth Measurement for controllers



Annex C. 3: TCP Attack Traffic Generation and Bandwidth Measurement for controllers

UDP Attack Traffic Generation and Delay (Jitter) Measurement for controllers



Annex C. 4: UDP Attack Traffic Generation and Jitter measurement for controller

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____