Intrusion Detection System in Software-Defined Networks

Vinicius Lopes Leite

Work oriented by:

Prof[a]. Dr. Tiago Miguel Ferreira Guimarães Pedrosa

Prof. Dr. Augusto Foronda

Prof. Nuno Gonçalves Rodrigues

Bragança, Portugal

2022/2023

Intrusion Detection System in Software-Defined Networks

Vinicius Lopes Leite

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master's Degree in Informatics under the scope of the Double Degree agreement with UTFPR.

Bragança, Portugal

2022/2023

Dedication

I dedicate this work to God, my family, friends and all people who were part of my life during this journey.

Acknowledgement

I would like to express my gratitude to all the professors from the Universidade Tecnologica Federal do Parana - Ponta Grossa and from Institute Polytechnic of Bragança for all the help throughout this academic journey, with all the opportunities and knowledge passed on to me over the last few years.

A Special thanks to my supervisors, Prof. Augusto Foronda, Prof. Tiago Pedrosa and Prof. Nuno Gonçalves, who inspired me to follow the academic and professional path I have taken, and giving invaluable help, without it, my research would have been impossible.

I would like to thanks the "CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)" research project.

Would like to thanks my family for all the support that they have provided for me during my entire life. My mom for always been there with love and wise words even don't understanding or knowing about my troubles, was always there for me. My dad for been a role model of a wise, successful, protective man which although been more inclusive, always provided the best he could for our family and protected us from everything. To my sister who taught me about independence, about chasing my dreams and never giving up independent of the obstacles that shows up.

Thanks to all the people that help during this last few years, a special thanks to special friends that i made during all this journey, Milton, Mendes, Jhony, Guth, Iury, David, Edson, Zé, Chuds, Nuno, Gabriel, Maicon, Fernando, Vinicius, each one of them marked my life in a unique way and have showed a great side of life.

Thanks to my sisters of another mother, Anne and Thaynara, which a long time ago

come to my life giving me support and always teaching me about love, friendship, so much more that i can even rememer.

For my beloved girlfriend Dalila, thank you for been my side in each difficult moment during the last year, for been my safe place during these journey and i would like to emphasize all the gratitude and love i have for you.

Abstract

Software-Defined Networking technologies represent a recent cutting-edge paradigm in network management, offering unprecedented flexibility and scalability. As the adoption of SDN continues to grow, so does the urgency of studying methods to enhance its security. It is the critical importance of understanding and fortifying SDN security, given its pivotal role in the modern digital ecosystem. With the ever-evolving threat landscape, research into innovative security measures is essential to ensure the integrity, confidentiality, and availability of network resources in this dynamic and transformative technology, ultimately safeguarding the reliability and functionality of our interconnected world. This research presents a novel approach to enhancing security in Software-Defined Networking through the development of an initial Intrusion Detection System. The IDS offers a scalable solution, facilitating the transmission and storage of network traffic with robust support for failure recovery across multiple nodes. Additionally, an innovative analysis module incorporates artificial intelligence (AI) to predict the nature of network traffic, effectively distinguishing between malicious and benign data. The system integrates a diverse range of technologies and tools, enabling the processing and analysis of network traffic data from PCAP files, thus contributing to the reinforcement of SDN security.

Keywords: Software Defined Network, IDS, Cybersecurity

Resumo

As tecnologias de Redes Definidas por Software representam um paradigma recente na gestão de redes, oferecendo flexibilidade e escalabilidade sem precedentes. À medida que a adoção de soluções SDN continuam a crescer, também aumenta a urgência de estudar métodos para melhorar a sua segurança. É de extrema importância compreender e fortalecer a segurança das SDN, dado o seu papel fundamental no ecossistema digital moderno. Com o cenário de ameaças em constante evolução, a investigação de medidas de segurança inovadoras é essencial para garantir a integridade, a confidencialidade e a disponibilidade dos recursos da rede nesta tecnologia dinâmica e transformadora. Esta investigação apresenta uma nova abordagem para melhorar a segurança nas redes definidas por software através do desenvolvimento de um sistema inicial de deteção de intrusões. O IDS oferece uma solução escalável, facilitando a transmissão e o armazenamento do tráfego de rede com suporte robusto para recuperação de falhas em vários nós. Além disso, um módulo de análise inovador incorpora inteligência artificial (IA) para prever a natureza do tráfego de rede, distinguindo efetivamente entre dados maliciosos e benignos. O sistema integra uma gama diversificada de tecnologias e ferramentas, permitindo o processamento e a análise de dados de tráfego de rede a partir de ficheiros PCAP, contribuindo assim para o reforço da segurança SDN.

Palavras-chave: Rede Definida por Software, IDS, Cibersegurança

Contents

List of Tables

List of Figures

Acronyms

AAA	Authentication, Authorization, Accounting
ACM	Association of Computing Machinery
AI	Artificial Intelligence
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
CIDN	Collaborative Intrusion Detection Network
CIRCL	Computer Incident Response Center Luxembourg
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial of Service
EC	Exclusion Criteria
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GB/s	Gigabytes per second
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HIDS	host-based intrusion detection system
HTML	Hypertext Markup Language

HTTP	Hypertext Transfer Protocol
IC	Inclusion Criteria
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPB	Polytechnic Institute of Bragança
IPS	Instrusion Prevention System
KB	KiloBytes
MAC	Media Access Control
ODL	Opendaylight controller
ONOS	Open Network Operating System
OVS	OpenVSwitch
PCAP	Packet Capture file
QoS	Quality of Service
RQ	Research Questions
SDL	Security Development Lifecycle
SDLC	Software Development Lifecycle
SDN	Software Defined Network
SSH	Secure Shell protocol
SSL	Secure Sockets Layer
SYN	Synchronize
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TTL	Time-to-live
UDP	User Datagram Protocol
VLAN	Virtual LAN
VM	Virtual Machine

Chapter 1

Introduction

The network architecture known as Software-Defined Networking (SDN) has arisen, in which the forwarding plane and control plane logic are separated. SDN is an innovative method for network programmability that is the capacity to use open interfaces and software to control, modify and manage network behavior dynamically as opposed to depending on closed boxes and proprietary defined interfaces [1]. The SDN architecture permits centralized management of the components of the data path apart from the technology employed by the network to link these devices that can come from various suppliers. All of the intelligence is embedded in the centralized control, which also keeps an overview of the data path elements and the links that link them across the network. The controller is suitable to carry out network management tasks and allows for simple modifications thanks to its centralized, up-to-date view [2].

In the current digital environment, research on security in SDN is crucial. SDN's novel architecture presents a number of distinct security challenges in addition to flexibility and scalability. Protecting vital network infrastructures requires an understanding of these issues and solutions. Because SDN is dynamic and centralized, research is needed to create strong security measures like access control policies, encryption protocols and intrusion detection and prevention systems. Research in SDN security is essential to keep ahead of new threats, such as DDoS attacks, illegal network access and data breaches, given the dynamic nature of the threat landscape [3].

Moreover, thorough research facilitates the creation of standards and best practices, which promote an SDN ecosystem that is more resilient and secure. In the end, the knowledge gathered from these kinds of studies is essential to guaranteeing the availability, confidentiality and integrity of network resources—a crucial component that supports modern digital networks' functionality and public trust [4].

1.1 Problem

Since the SDN technologies are a recent work, the lack of traditional physical network boundaries in SDN necessitates robust security measures at both the controller and switch levels to protect against malicious activities. So, the main challenge is to develop a intrusion detection system, that can identify and prevent the actions of malicious activities inside a SDN network.

1.2 Goals

The main goal that this work tries to achieve is the development of a efficient IDS for the SDN scenarios, which need to work in a near real time processing to prevent malicious activities.

This research starts with a study of the recent scientific works in the SDN scenario and the implementation of security measures for this type of networks, making a robust literature review for this work. The next step deploys a scenario for the SDNs based on the more used frameworks in the current market, collect and analysis of traffic will be performed to detect malicious activities coming from any host. With the analysis done, the process goes to a development of a AI model capable of identify these attacks and warn the administrator of the networks safety.

1.3 Structure of Document

This document is structured in six chapters and 5 appendix. Chapter 1 refers to the introduction, the problem description and the objectives of this dissertation. Chapter 2 explains how the literature research was made, the main topics and keywords that were searched and the result of the gathering of information. Chapter 3 bring the approach defined to solve the problem. Chapter 4 explain the implementation of this work and how the tools defined in chapter 2 were used. Chapter 5 presents every experiment executed in order to evaluate and improve the proposed system. Lastly, chapter 6 brings the final conclusion of the discussion and future work.

1.4 Acknowledgement

This work was developed within the "CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)" research project.

Chapter 2

State of the art

This chapter is arranged into four sections, the first will explain the main concepts of this work, the second explains the methodological approach used in the systematic literature review. The third one brings a overview of the research papers selected in the second section with the purpose of contextualize the main discussion and problems that the scientific community has identified during the last years. Then a final section with the explanation about the tools that were chosen for this work.

2.1 Concepts

To have a fundamental comprehension of this work it is important to understand the main concepts. In this section, we will present an overview of the ideas used in the subject and explain them in an accessible and understandable manner.

2.1.1 Software Defined Network

Feamster, Rexford, Zegura et al. [5] explain that Software Defined Networking or SDN, is a new approach that has been studied during the past years that consists in a networking strategy that separates the control plane from the data plane. The network control functions are replaced from the individual networking devices to a centralized controller,

which makes it easier to manage and configure the network as a whole as it can be see in figure 2.1.

The centralized architecture from the controllers can communicate to the applications through the northbound interface which interact with the controller and request specific network policies or configurations. This interface provides a standard API (Application Programming Interface) for applications to do this communication with the controller. And in order to configure network communications for switches or routers, the controller uses a southbound interfaces to program and operate network devices.

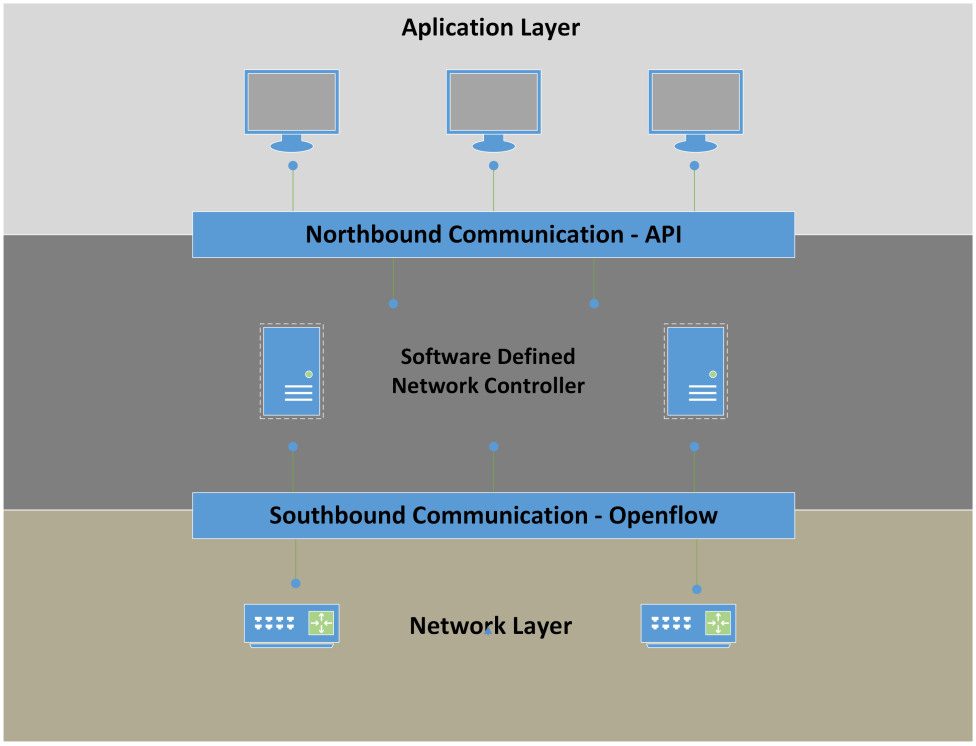


Figure 2.1: Software Defined Network Structure

Feamster, Rexford, Zegura et al. [5] explains that to enable the idea of the SDN, the OpenFlow protocol is the best option to help it, which provides a standardized way for the controller to communicate with the network devices. Without a standardized protocol like OpenFlow, it would be much more difficult to create a centralized controller that can interact with a wide variety of network devices.

Since OpenFlow allows the controller to configure network devices in real-time,

network behavior may be fast modified in response to changing network conditions. This might help to guarantee that apps obtain the resources they require to run effectively without overloaded the network or causing performance difficulties.

Malishevskiy, Gurkan, Dane, Narisetty, Narayan and Bailey et al.[6] Shows that SDN and OpenFlow have several benefits over traditional networking methods. SDN and OpenFlow give improved network visibility in addition to the benefits of being flexible and programmable. The controller has complete insight into network traffic and may use it to make better judgments about how to forward packets. This is especially useful in contexts with unpredictable network traffic, such as data centers or cloud computing environments.[6]

Another benefit of SDN and OpenFlow is that they simplify network administration. Traditional networking requires network managers to setup each device separately, which may be time-consuming and error-prone. The centralized controller can automate various network administration functions, such as routing and load balancing, using SDN and OpenFlow, making it easier for administrators to operate the network and reducing the chance of mistakes.

2.2 Literature Review Methodology

This section explain the methods of data collection and analysis, the tools used to find and organize the information.

To achieve a solution to the problems described in section 1, a research among the recent works made in the area was needed in order to reach a better perception of the problems, solutions and main concerns in this field. Also in this section some context will be presented to define concepts that are crucial to the understand of the work.

Three main research questions were defined in order to help the gathering of data to develop this work.

- **RQ1:** What are the main SDN controllers that are been used?

- **RQ2:** What are the SDN threats/attacks?

- **RQ3:** What are the security mechanisms and countermeasures to mitigate SDN Intrusion?

Trying to find the best answers to the Research Questions, it was necessary to select keywords and some synonyms that combined with logical operators formed a search string to be used in the Researches repositories.

Keywords	**Synonyms**
Software Defined Network Intrusion Detection System Intrusion	'SDN'; 'Software-Defined-Network'; 'SDN Controller'; 'Detection'; 'IDS'; 'Monitoring'; 'Attack'; 'Infection'; 'Threats'; 'Threat';

Table 2.1: Keywords and Synonyms for this work

Search String
("Software Defined Network" OR "SDN Controller" OR "Software-Defined-Network") AND ("Intrusion" OR "Attack" OR "Infection" OR "Threats" OR "Threat") AND ("Intrusion Detection System" OR "Detection" OR "IDS" OR "Monitoring")

Table 2.2: Search String based on the keywords

2.2.1 Data Collection

With the search string set, the data collection and organization were carried through the website Parsifal (www.parsif.al), because of it's organization method and easy understanding of the interface to organize and due to the ways that you can choose to classify the studies made. Inside the website multiple digital libraries were selected to proceed with this part.

Using the search string through all the digital research libraries, were reached a total of 3147 scientific studies in the area during the last six years.

Digital Research Libraries	URL
ACM Digital Library	http://portal.acm.org
IEEE Digital Library	http://ieeexplore.ieee.org
Scopus	http://www.scopus.com

Table 2.3: Digital Libraries

Digital Research Libraries	Total Results
ACM Digital Library	1016
IEEE Digital Library	651
Scopus	1480
Total	3.147

Table 2.4: Results of the Libraries

2.2.2 Analysis and Selection Phase

During the analysis and selection phase of the scientific studies, inclusion and exclusion criteria were utilized to separate studies that do not fit this research, in order to find relevant studies for the review.

The inclusion criteria outline the characteristics that research must possess or discuss in order to be taken into consideration for the review. In this research the inclusion criteria were:

- **IC1:** Papers that are focused on build an SDN environment or SDN applications.
- **IC2:** Papers that are focused on security area in SDN.
- **IC3:** Papers that bring threats and solutions to SDN problems.
- **IC4:** Papers that have similar approaches or solutions are compared and the more recent and complete are included.
- **IC4:** Papers that include data sets for testing.

The exclusion criteria focus on classify research that approach field of area that were not been discussed in the work. In this research the exclusion criteria were:

- **EC1:** Papers that focus on implementing SDN in Smart Cities or smart vehicles.

- **EC2:** Papers that uses SDN in their approaches but aren't focused on it's security or implementation.
- **EC3:** Papers that were outdated in their approach.
- **EC4:** Papers that solves the same problems were excluded leaving only the most recent or complete research.
- **EC5:** Papers that focused in solving industrial problems and not SDN problems.

With this methodology is possible to classify the papers in four distinct categories:

- **Duplicated:** Research papers that already were presented in a previous digital library
- **Rejected:** Research papers that do not fit the purpose of this work or use similar approaches that more completed papers.
- **Accepted:** Research papers that could help this work in a significant way

The main criteria for to choose. reduce and classify this papers were:

- **Cibersecurity detection:** Research papers that show different types of security approaches to detect malicious activities.
- **Cybersecurity in SDN:** Research papers that shows approaches of cybersecurity in SDN scenarios
- **Different tools to implement an SDN scenario:** Research papers that use different technologies to deploy the SDN scenario like ODL, ONOS, Ryu. etc...
- **Datasets for malicious activities:** Research papers that use Malicious dataset to train an AI model to detect attacks.

The results of the analysis can be see in the table 2.5, all the papers classified as accepted were analyzed for building the research of this work.

Classification	Total Results
Duplicated	687
Rejected	2.274
Accepted	186

Table 2.5: Classification Results

2.3 Literature review

Moussaid, Nadya and Toumanari et al. [7] affirms that besides the benefits of SDN, the controllers also faces security challenges in different areas of its architecture. The use of APIs can introduce vulnerabilities by facilitating communication between different layers of the network architecture. If they are not properly verified and protected, attackers or malicious user may exploit their vulnerabilities at various tiers, potentially leading to unauthorized access to the network infrastructure or the ability for malicious actors to manipulate network traffic. Additionally, the insertion of malicious code into unprotected APIs can result in significant harm, such as data theft, network disruption, or even complete network compromise. Therefore, it is crucial to ensure that SDN APIs are adequately secured and protected against potential security threats.

One of the biggest issues presented by Zhu, Wang, Li et al.[8] is about the risk of an unauthorized access, because of the fact that the controller runs a role as the brain of the network. This level of control could allow an attacker to intercept, modify, or redirect network traffic, leading to data theft or network downtime. In addition, an attacker who has access to the SDN controller can potentially manipulate the network's routing decisions, leading to network congestion or even network-wide failure.

The SDN switches issues are primarily related to the vulnerability of Flow table entries limitation as informed by Birkinshaw, Celyn and Rouka et al. [9], which can make them extremely sensitive to Distributed Denial of Service (DDoS) attacks. These attacks can overwhelm the switches by flooding them with traffic and exhausting their resources, ultimately causing the network to fail. In addition to DDoS attacks, other security threats that exploit the vulnerability of Flow table entries in SDN switches include buffer overflow

attacks, packet injection attacks and malware injection attacks.

If any of these attacks are successful the entire network are going to be in danger. Thinking about solving and avoid these issues the development of effective security measures, such as traffic filtering, access control, and anomaly detection, are vital to prevent and mitigate the impact of these attacks. Techniques like monitoring and analysis of network traffic patterns can help identify potential security threats and enable proactive measures to be taken to enhance network security.

On a initial phase of the study at the area of SDN controllers, Arbettu, Khondoker, Beyarou et al. [10] analyzed four controllers, two of which are Java-based open source controllers: OpenDaylight (ODL) and Open Network Operating System (ONOS), and two others are Ryu, a widely used Python-based controller in research areas, and Rosemary, a proprietary controller software. Each controller was analyzed to assess the security of its interfaces, processes, and internal data.

To evaluate the security levels of the individual controllers, the STRIDE threat modeling framework was used, which was developed by Microsoft for threat modeling purposes and has been an integral part of the Security Development Lifecycle (SDL). In the SDL approach, security of an application is prioritized over other parameters such as performance, availability, and integrity for evaluation during the Software Development Lifecycle (SDLC). The STRIDE framework is an acronym for six threat categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privileges. The author created an individual STRIDE threat matrix for each of the controllers to summarize their maturity against known attack categories.

Arbettu, Khondoker, Beyarou et al. [10] reached a result that all controller are prone to threats or vulnerabilities. Some secure modes available in a few controllers are optional to maintain higher performance levels. The ONOS controller is susceptible to known security threats, but the ONOS team already started addressing these issues. The grate disadvantage of Ryu controller is the multi-threaded architecture that leads in the DoS attacks due to the lack of authorization for adding flow rules and event consumption. Rosemary for the other side employs an effective resource access control

mechanism, creating multiple micro-NOS instances for every application is an overhead on performance. OpenDaylight surpassed the other controllers for several reasons, like modular structure, constant monitored security and it's own module to support Denial of service attacks.

Mat Isa, Mhamdi et al. [11] proposes a new approach to enhance network security by combining Software-Defined Networking (SDN) and machine learning techniques. The proposed intelligent security framework leverages the centralized control plane of Ryu SDN controller to facilitate efficient network monitoring and anomaly detection. By combining deep learning algorithms with SDN, the framework enables real-time network traffic analysis and anomaly detection.

The architecture for the solution was composed in two layers, the first is the gathering data on network traffic within the SDN framework to inform the controller's decision-making process. To achieve this, a standardized Openflow protocol is employed, which regularly dispatches Openflow flow stats request messages to all switches connected via Openflow. These switches will then respond to the message with traffic flow statistics. The collected data is then collated and utilized as required input for the controller to execute appropriate actions.

The next tier of the system features an intrusion detection algorithm that combines the use of an auto encoder and random forest algorithm. The collected data is then formatted to enable the controller to execute detection and prevention functions within the adaptive machine learning module. It is critical for the detection process to be quick and accurate, given the high-speed flow of data. Following the classification, normal traffic will be allowed to pass uninterrupted, while anomalous traffic will face penalties.

For the development of the machine learning, the usage of the KDD-99 dataset [12], presented by the table 2.6.

Dataset	**Normal**	**DoS**	**Probe**	**R2L**	**U2R**	**SUM**
Train	67,343	45,927	11,656	995	52	125,973
Test	9,711	7,458	2,754	2,421	200	22,544

Table 2.6: Total packets in the dataset

The approach achieves high accuracy of 98.4% for intrusion detection while reducing the time needed for training and execution. The results demonstrate that the model is efficient for real-time intrusion detection, making it a promising solution for improving network security in SDN-based intrusion mitigation architecture.

In a distinct approach Majid Ali and Pervez et al. [13] describes the use of metasploit and Kali Linux to simulate seven different types of attacks within an SDN environment. The attacks include DDoS, SSH, FTP, HTTP, ICMP, ARP, and Scan attacks. In order to detect these attacks, the author proposes a solution that combines collaborative intrusion detection and blockchain technology. The solution uses Snort signature-based detection and deploys an SDN-based test-bed that uses three collaborated Snort IDS. These IDS nodes securely receive new signature updates from the SDN Ryu controller, allowing for effective sharing of important information.

Collaborative intrusion detection systems typically work by sharing information about network traffic patterns, anomalies, and potential security threats. With that in mind the block-chain method was chosen by it's purpose of using the decentralised architecture to keep information safe against any threat that can adulterate them. The information exchange between IDS enables to identify breaches in the security.

The experiment consist in 2 Virtual Machines (VM), where VM-1 system is designed through the integration of SDN Ryu and Snort. The SDN controller is responsible for programming the network operation, including updating Snort's signature rules and data-plane flow. This program is triggered when Ryu receives the Packet in message from OpenFlow. To modify the data-plane of the proposed CIDN network, the Ryu controller employs Link-A in this virtual machine, while Link-B is used for Snort signature sharing towards the CIDN network. VM2 represents the network domain, where a Mininet emulator is used to create a CIDN virtual network with Mininet simulator. The CIDN network comprises three Snort IDS nodes, all of which are deployed with default signature rules. When a new packet with malicious attributes arrives, the main HIDS Snort node receives new signature updates from Ryu via Link-D. These updates are then disseminated to all other CIDN nodes via blockchain.

The collaborative approach of this solution is found to generate very few false alerts, with an average of 5% False Positive rate and False Negative rate for DDoS and HTTP attacks. However, the average False Positive rate and False Negative rate for other attack types were nearly 10%, which can be attributed to each IDS detecting burst attack input traffic. Despite this, the proposed solution shows promising results in terms of detection accuracy and performance. By utilizing the Ryu Controller and blockchain technology, the solution provides a verifiable information sharing. The article highlights the importance of collaborative intrusion detection approaches in SDN environments, particularly in safeguarding against various malicious attacks.

Also using snort IPS system Pratama, Fauzan and Suwastika et al. [14] describes the use in a Ryu controller as well but using a different approach analyzing the packets through a fuzzy logic system, where the validation process for the fuzzy design involves determining the block's duration.

Using SDN technology to process any packets present in the network, with Ryu handling the initial processing and Snort analyzing the packets to determine their level of danger based on established rules. If a packet is deemed safe, Snort will send it back to Ryu for forwarding to the intended destination. In the other hand, if the analysis result that the packet is malicious, the process of mitigation and alert will start. Generating warnings to the controller, and blocking the passage of the packet for an amount of time. As a result of these actions, the packet will not be delivered to its destination host, and the sender host will be prohibited from sending any further packets for a period commensurate with the attack frequency.

This is achieved by designing the fuzzy logic system 10 times, analyzing each design, and then selecting the most appropriate one for use in the system. Based on the outcome of the various trials, the SDN infrastructure that has been furnished with a flexible IPS has the capacity to identify cyber assaults and restrict the aggressor's host for a duration that depends on the frequency and category of the executed attacks. Furthermore, the implementation of the fuzzy algorithm added an execution time of 0.228 milliseconds to the process.

Trying to understand more about the difference between the security problems in the SDN controllers, Badotra, Sumit and Panda et al. **snortDDoS** brings a similar approach on the use of the Snort system to reach an DDoS detection system on the ODL and ONOS controllers.

In order to emulate a topology for the tests, Mininet tool was used to create and distribute the Virtual machines, and for the Denial of Service attack since ODL and ONOS uses TCP port number 8181 for HTTP, it can be attacked for both HTTP and TCP SYN Flood attack, so four different tools:

- **Xerxes:** DoS tool that specializes in executing DoS attacks by launching multiple independent attacks against numerous target websites.
- **Tor's Hammer:** This tool is a DoS application that employs a low-rate hypertext transfer protocol POST (OSI Layer 7) to carry out its attack. It utilizes a classic slow POST approach, where the HTML POST fields are transmitted slowly under a single session, in order to execute the DoS attack..
- **Hping3:** A command-line tool and packet analyzer that is capable of working with a range of TCP/IP packets. It has support for protocols such as UDP, ICMP, and RAW-IP, and includes a trace route mode. Hping3 also offers features such as secure file transfer and numerous other options to facilitate more advanced functionality.
- **Nping:** A versatile tool that can create network packets for a wide range of protocols, giving users complete control over the protocol headers. While it is commonly used as a simple ping utility for detecting active hosts, Nping can also function as a raw packet generator for stress-testing network stacks, carrying out ARP poisoning, launching DoS attacks, and tracing routes, among other applications

The results of the DDoS test revealed that ODL detected the attack faster than ONOS and took longer to go down. To detect DDoS attacks, local SDN DDoS alert rules were configured using various SNORT rules. The alert rules were created by setting the source

traffic from any network or port, and if the traffic was coming to the SDN controller at TCP Port Number 8181, then the message was marked as an SDN connection attempt from an outside network.

The detection time and packet loss were recorded for various scenarios. The number of packets bombarded affected the time when the controllers went down, and the overall network functionality stopped. Using this approach it has been observed that ODL detects DDoS attacks in a shorter time than ONOS and experiences a delayed response compared to ONOS.

In another approach to DDoS detect and mitigation system, Cajas and Budanov et al. **DDoSOpen** uses an application in Java and choosing OpenDaylight as the controller. The fundamental concept is that the software computes the data transfer rate of various end-to-end connections that are set up in the network. This is accomplished through the deployment of forwarding regulations that permit the accurate transmission of data from a beginning location to a termination location and by amassing statistical data regarding these rules.

The forwarding rules consist of matching fields, such as the source IP address, destination IP address, and input port. The action of these rules is to forward packets to a specific output port. When the matching fields of incoming packets perfectly match those of a forwarding rule, the rule is applied, and the number of bytes for that rule is updated in the switches. The application also stores the number of bytes for each rule, along with a tuple consisting of the switch identifier, input port, destination IP address, and destination port. This allows the forwarding rules to be identified across multiple switches and connections.

By gathering information from all switches in the SDN network, the application can make an estimate of the throughput for each end-to-end connection. The estimation is periodically updated based on a time interval, referred to as the "refreshing_time" variable, which is set to 5 seconds. This value is chosen to balance the need for timely detection and mitigation of DoS attacks with the need for accurate estimation. If the throughput of any connection exceeds a threshold, the application will identify the

corresponding tuple and end-to-end connection, and install new blocking rules not only on the switch responsible for the throughput estimation but on all switches along the end-to-end path. These blocking rules will have the same matching fields as the forwarding rules but their actions will be to drop packets, effectively blocking the end-to-end connection.

Understanding more about what are the main open source SDN controllers in the actual market, Ohri, Pulkit and Neogi et al. [15] considers that are more than 20 SDN Controllers available in the market, and ONOS and ODL are the most popular and complete in the actual scenario, presenting a study about how these SDNs can face DDoS security threats.

In OpenDaylight the use of the feature Defense4All is the main module of defense in DDoS attacks and other security components include Authentication, Authorization, accounting (AAA), Controller Shield, ODL-Cardinal, Secure Network Bootstrapping Interface, and more. Defense4All divide it's functionalities in two parts, Detection and Mitigation part. In detection, Defense4All monitors network traffic in order to find any suspicious behaviour of any host connected to the network managed by the controller, using traffic flows in the Openflow Switches. After analyzing the network traffic, Defense4All creates different baseline metrics, such as the mean number of users connecting to the server, the number of responses and replies sent back and forth, and others. If the system detects a consistent and significant deviation from this established baseline, it will send an immediate alert to the ODL Controller to signal the presence of suspicious activity. To mitigate the attack the feature uses the Attack Mitigating Hosts Technique redirecting the malicious traffic to a temporary host mitigating the suspicious traffic from affecting the network. It makes the hosts return to their normal flow of operation.

By the ONOS controller side, it has an application layer access control feature which brings a defense from malicious applications, allowing applications to use the most part of the resources in the network. ONOS also has a Security mode which provides two main features, Application authentication and permission based access, protecting the network

from malicious users.

In the paper, Ohri, Pulkit and Neogi et al. [15] divides his tests in DDoS attacks in three types: Low Rate, Medium rate and High rate attacks. Testing the ODL feature, Defense4all worked perfectly in low rate attacks, redirecting the malicious traffic to a temporary host and keeping the response time of all hosts in the same time as always. Now in the medium rate, the feature didn't work so well, the nodes that were attacked had they packet loss rate in 60% and losing response time. And in high rate DDoS attacks ODL failed again in stop the attack having around 87% packet loss and a significant deterioration in the response time again. But in a more positive side, when the ODL finally realize that it is under attack, all operations are stopped, restarted and starts to work again in a normal mode.

In the ONOS controller since it does not have any dedicated anti-DDoS mechanism, it fails to work even for a low-rate DDoS attack. The fist approach was performing a host to host attack in which two hosts, were used to attack a node, the network connectivity pings performed well during this phase but after the SYN Flood attack the ping from the nodes started to becomes unreachable. The ping between all ten nodes becomes unreachable. In addition, the ONOS GUI also stopped responding. Which demonstrates a complete failure of the controller against a simple SYN Flood DDoS attack.

In a practical scenario, attackers create a group of botnets to target a server that responds to numerous hosts. However, in the experiment conducted by the author, only two hosts were utilized to generate attack traffic, but it did not succeed. Therefore, it can be concluded that ONOS Controller is incapable of defending against any form of DoS or DDoS attack in the real world. In contrast to OpenDaylight, ONOS Controller does not revert to its normal operation flow once the attack is halted, and the node pings continue to show "Destination Host Unreachable". The controller only resumes its normal function after being manually restarted.

The problem caused by DDoS attacks is a well-known issue that affects SDN controller and both ODL and ONOS suffer from this attacks. ODL has attempted to fix this issue by implementing an optional feature that consist in a custom anti-DDoS mechanism

called Defense4All, but it can only mitigate low-rate DDoS attacks and is ineffective against medium and high-rate attacks. In contrast, ONOS has no anti-DDoS mechanism and cannot handle low-rate DDoS attacks. Ohri, Pulkit and Neogi et al. [15] give more emphasis in the need for SDN developers to address these security concerns and highlights that ODL is more resistant to DDoS attacks than ONOS.

2.4 Related work

In the work developed by Olivera and Pedrosa et al.[16] was proposed an scalable architecture that could handle large amount of data generated by network traffic using a variety of current technologies and also develop an DGA detector in the traffic captured in the network, which fits perfectly in this work's scenario.

Using an strategy of combining Apache tools to create an scalable scenario where it's possible transmit all the data produced in a SDN network through a scalable queuing module been received an stored in scalable persistent storage for backups and future checkups, and a scalable analysis module to process all the data received. These technologies will be present in the next section where the tools used in this work are introduced

The AI usage in this work, brings an approach using the TensorFlow. a Google-developed open-source machine learning framework, for creating and refining deep learning models, it is commonly utilized. TensorFlow offers a flexible ecosystem of tools and packages that allow programmers to quickly produce a range of deep learning and machine learning models.

The machine learning algorithm chosen to this research was the Long-short-term memory neural network architecture, where it excels in applications like natural language processing, speech recognition, and time series forecasting where understanding the context and temporal relationships in data is essential for precise predictions. They are particularly good at capturing dependencies and patterns within sequences.

The results deployed by this research demonstrate a promising evolution in the usage

of AI in a near real-time analysis where the system has a precision and a recall of 98% being a reliable detection model.

Based on this approach we will adapt the scalable solution to fit in a SDN scenario, developing a system where an SDN solution could work and use scalable system as an Intrusion detection system.

2.5 Tools

After the analysis of the works done in the SDN area, the tools chosen to be used in the work were selected and in this section the functionalities and the reasons of why each of them were chosen are explained.

2.5.1 OpenDaylight

With more than 20 SDN controllers that are under constant development in the actual market scenario, Badavaro and Constantino et al. [17] ONOS and ODL are the most popular and complete tools that are been used for giants network companies like Huawei and Cisco. The chosen controller for this work is Opendaylight based on the leak of recent works in the scientific community about it and because of it is the recommended option for a new generation of Cisco switches [18].

OpenDaylight employs open protocols to provide centralized, programmatic control and network device monitoring. The controller provides an interface for connecting network devices quickly and intelligently to achieve optimal network performance, similar to how an operating system provides an interface for connecting devices to a computer. OpenDaylight is a collaborative open-source project hosted by the Linux Foundation, with the primary goal of accelerating the adoption of SDN and creating a solid foundation for NFV **ODLmanual**.

The project is governed by an open community decision-making process, which has brought together community developers and open-source code to achieve this goal. OpenDaylight can be a core component of any SDN architecture, providing the ability

to minimize operational complexity and extending the life of existing infrastructure while enabling new services and capabilities. The architecture of OpenDaylight is multi-layered, with the controller platform being the main layer. The OpenDaylight Controller, which acts as the brain of the network, resides in this layer and manages traffic flow from switches using flow tables. The controller can be executed on any operating system as long as it supports Java. Multiple protocols can be supported on the southbound, such as OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc.

2.5.2 OpenVSwitch

Since the physical switches that allow the use of the OpenFlow protocol are respectively new and more expensive than the usual ones, we opted for an alternative to conventional hardware-based network switches that is offered by Open vSwitch (OVS) tool, an open-source software switch. It was developed to overcome the shortcomings of conventional network switches and to make it possible for SDN environments to support more flexible and dynamic network administration.

OVS can be utilized as an OpenFlow switch, offering a programmable and flexible forwarding engine that can be managed by a centralized SDN controller. The OpenFlow protocol is implemented in OVS's datapath, allowing it to receive and process OpenFlow messages from the controller. The controller may configure the forwarding tables in OVS using OpenFlow, describing how incoming packets should be forwarded depending on parameters such as source and destination addresses, protocol types, and Quality of Service (QoS) needs. Furthermore, the OpenFlow integration enables OVS to interact easily with other OpenFlow switches, allowing the establishment of large-scale, heterogeneous SDN networks that can be controlled and managed centrally. **OVSmanual**

We could see in the studies presented in the previous section, that the controller was implemented in a host and all the network that it manages were a virtual network simulated by the mininet tool or other network virtualization tool. And inside this

virtualized networks the switching process are build with OVS tool. It employs OVS as its default switch implementation, allowing users to construct and manage virtual switches, bridges, and routers that may be dynamically configured and maintained based on network requirements. Because of its versatility and programmable, OVS is well-suited for usage in Mininet, allowing users to design complicated network topology and test different network configurations in a controlled and isolated environment. [19]

Other well known tool analyzed in these studies is the Proxmox Server Virtualization Software, which the default virtual switch implementation is OVS, which provides users with a versatile and configurable software switch for creating and managing virtual networks. OVS is well-suited for usage in Proxmox due to its compatibility with a broad range of SDN controllers and network functions, allowing users to design and administer virtual networks that can be dynamically built and maintained based on the demands of the virtual environment. [20]

2.5.3 PCAP file

A PCAP file is a type of file that stores network traffic captures. Farrukh, Khan, Irfan and Wali et al.[21] defines the usage of this type of files for network analysis, troubleshooting, and security purpose. The PCAP files are used to capture packets that are been transported on the network. The PCAP file uses a binary format of the packets that were captured, and including information about their headers, payloads, and metadata. The packet header bring some of the most important characteristics about the packet, like the size, date, and protocol that was used. This data may be used to study network traffic and obtain insight into network applications and protocols behavior.

The PCAP file is one of the most used formats by a grant variety of tools, because of the easy architecture (figure 2.2) to extract information about network traffic, IP addresses of the senders and receivers, packet contents. The data can be used to solve and monitor network faults, identify security concerns, and improve network performance.

They also can be used to reconstruct network traffic after any incident or attack, allowing investigators to follow the trace back to the cause of a security breach or other network-related issue.

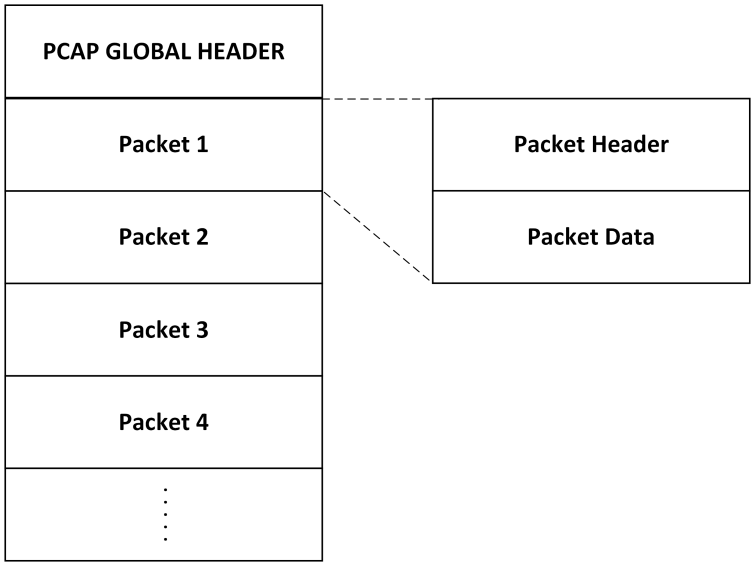


Figure 2.2: PCAP File Structure

2.5.4 Kafka

Peddireddy et al. [22] thinking about the large amount of data that would be necessary to transmit and store in a traffic analyze, the best option would be a distributed and scalable tool, which the Apache Kafka fits in the exact parameters the were judged as necessary. The tool works in a distributed way, handling a real-time messaging system that can care large volumes of data from multiple sources.

Kafka is made up of three major divisions: producers, customers and brokers. Producers take care about the service of creating data and publishing it to the Kafka cluster; Consumers are in charge of get this data and processing that data; And brokers manages of all data that reaches the Kafka cluster.

Kafka utilizes an approach which consists is categorize the data into topics and the producers sends messages specifying the topic with the data. Consumers can choose in which topics it desires to receive real-time messages. Kafka also supports data storage

and processing, allowing for more efficient processing of massive amounts of data.

The architecture is intended to be scalable and to accommodate failures. Data partitioning is used to ensure redundancy among the numerous brokers in the Kafka cluster and each partition is duplicated across these brokers. Kafka also enables for management changes in the cluster's brokers without interfering with data processing. Kafka's high throughput and low latency make it ideal for real-time analysis, log aggregation, and data streaming.

Kafka allows external systems to connect and use the services, and it's APIs are available in a variety of programming languages, making it accessible to a grant variety of developers. [23]

2.5.5 Hadoop Distributed File System

Since we are searching for a scalable and secure way to handle all the data, with the Kafka it's possible to transmit large amount of data, and in order to store it all in a secure and a reliable way , it will be necessary a tool that can provide these services.

The Apache Hadoop framework is a open source tool created to handle huge data set processing and analysis in a distributed computing environment. Providing scalable and fault-tolerant platform for storing, processing, and analyzing large volumes of data across clusters.

Among different services provided by the framework, Tian, Yu. et al. [24] defines the Hadoop Distributed File System (HDFS) as an interesting tool that delivers storing and managing ways to large volumes of data across multiple nodes in a Hadoop cluster. Data is kept in files, which are subsequently separated into blocks. These chunks are then dispersed among several cluster nodes. Each block is duplicated across numerous nodes to achieve redundancy and fault tolerance.

In addition a big advantage of using the Hadoop framework in this work, is the capability communicating with other Apache technologies such as Kafka. Kafka can be used to receive and process streaming data in real-time, while HDFS can be used to

store and process huge amount of data. This interaction between the tools can turns possible to handle real-time data processing [25].

2.5.6 Capture tools

During the approach and implementation chapters it will be discussed why these methods of capture and this tools where chosen to this work. The tools were the TCP Dump and python3 language to develop the scripts of capture.

Python3 is the most recent version of the programming language Python. It is a well-known and frequently used language noted for its ease of use, readability and adaptability. Python 3 has a large number of libraries and frameworks that make it suited for a wide range of applications such as web development, data analysis, machine learning and automation. It features a simple and straightforward syntax, making it simple to understand and produce code. Python 3 also stresses code readability and encourages the usage of best practices in programming. Its vast community support and extensive standard library make it a useful tool for developers to construct sturdy and efficient applications. The libraries and how it was used will be explained in the Implementation section 4.4 [26].

TCP Dump is a network monitoring and analysis program that uses command-line packet sniffing techniques. It records network traffic in real time and offers precise information on packets passing through a network interface. TCP Dump allows users to create filters to capture certain types of traffic depending on protocols, source/destination IP addresses, ports, and other parameters. It contains a plethora of information, including packet headers, payload contents, source/destination information and time information. Also in the section 4.4 will be explained the usage of this tool **TCPdump**.

2.5.7 IA analisys

H. Kang, H. Kim et al.[27] define the use of artificial intelligence a major differential for the future of IDS, redefining the way we keep our digital ecosystems safe from potential attackers. Traditional IDS systems often struggled to keep pace with the dynamic and increasingly sophisticated nature of modern cyberattack bringing an important role to the AI usage because of its revolutionary characteristics like the ability to adapt, learn and respond in real-time to a large amount of threats or events. AI succeeds in this difficult environment by continuously analyzing massive data sets, spotting trends and foreseeing possible dangers before they materialize into full-fledged attacks.

The capacity of AI-powered IDS to identify anomalies and deviations from normal network activity, which frequently go unnoticed by human operators, is one of their primary advantages as explained by Zebin, Tahmina, Rezvy, Shahadate, Luo, Yuan et al.[28]. Faster response times as a result of the improved detection accuracy significantly reduce the vulnerability window and reduce potential impact. Furthermore, AI has the ability to automatically prioritize alarms, ensuring that security professionals focus their efforts on the most serious threats. The overall cybersecurity stance is greatly strengthened by this resource allocation optimization.

The ability of AI to change and advance is also quite valuable. Randhir, Prabhat, Rakesh , P. Gupta, Garg, Hassan et al.[29] shows that new attack pathways and methods, it can hone its detection capabilities over time, ensuring that IDS systems are resilient to new threats. This adaptability extends to the scalability of AI-driven IDS, which can easily manage the rising volume and complexity of data flow in the connected world of today.

In addition to threat detection, AI equips IDS systems with powerful response mechanisms so they can respond instantly and automatically to reduce risks. In addition to reducing human involvement, this quickens the incident response procedure, which is essential for avoiding data breaches and limiting potential harm.

As it will be possible to see in the chapter 4, we opted by using the Random Forest

Algorithm after getting the results of the performance presented by the table 4.7. We got a satisfying performance from almost every algorithm based on the training using our malicious traffic dataset. Afroz, Islam, Rafa, Samin and Maheen et al. [30] defines it as an ensemble learning technique called Random Forest builds numerous decision trees, each of which is based on a unique random sub sample of the initial training data. It adds two layers of randomization during tree construction: first, by choosing only a portion of features at each node split and second, by allowing the trees to develop naturally without being pruned. This group of decision trees uses majority voting for classification tasks or average forecasts for regression tasks to combine their individual predictions and tap into their collective wisdom. With this method, over fitting is reduced, model generalization is improved, and outliers and noisy data are robustly handled. Because it can take advantage of unpredictability at several stages of the modeling process, Random Forest is effective and frequently used.

2.5.8 Passive DNS

An important cybersecurity and network monitoring tactic is passive DNS, which focuses on gathering and archiving past DNS information without actively querying or tampering with the DNS infrastructure. Network traffic is continuously observed in a passive DNS system, and information about DNS queries and answers is gathered and examined. With the help of this method, organizations can track domain-related activities, investigate and analyze network events, and improve security thanks to the priceless archive of historical DNS data it offers [31].

Particularly helpful for a range of security and research needs is passive DNS data. It facilitates the identification of malicious activity by exposing DNS request patterns linked to malware, phishing, botnets, and other cyberthreats. Additionally, it helps with threat intelligence, enabling businesses to proactively find IP addresses and domains that might be malicious. Passive DNS can also be a very useful tool for forensic investigations and incident response, assisting security experts in piecing together the timeline of events that

led up to a cyber incident. Passive DNS helps with network security, threat detection, and cyberattack mitigation by providing a history of DNS resolutions [32].

In this work we count with an access to the CIRCL database [33], our partners in the Passive DNS approach, which has released to us the use of their PyPDNS python library which provide the queries to consult their database and use the informations in order to develop a Passive DNS model that can identify potential malicious DNS.

2.5.9 PyPacker

Pypacker [34] is a python library that the main object are packet capturing and network packet manipulation. It offers a range of tools and features that enable users to work with, create, break down, and examine network packets at a low level. For example, packet dissection lets users break down and examine network packets in order to extract different packet attributes, including protocol information, source and destination IP addresses, port numbers, and payload data. This makes it simple to analyze and work with packets using these protocols, which is helpful for tasks like examining network traffic and extracting particular data from packets. This is especially helpful for complicated networking tasks involving several protocol layers.

2.5.10 Sci-kit Learn

A popular and potent Python machine learning library is called Scikit-Learn [35]. For data scientists and machine learning practitioners, it is a priceless resource because it provides a wide range of tools and functions for different aspects of machine learning. For a wide range of machine learning applications, including dimensionality reduction, clustering, regression, and classification, Scikit-Learn offers a standardized and intuitive user interface. Its easy-to-use and thoroughly documented API makes creating, testing, and assessing machine learning models more straightforward.

Scikit-Learn's dedication to performance and efficiency is one of its main advantages; the majority of its algorithms are written in low-level languages like C, which guarantees

fast computation. Additionally, it facilitates model selection, hyperparameter tuning, and data preprocessing, forming a complete ecosystem for creating end-to-end machine learning solutions. In addition to having a large selection of machine learning algorithms, Scikit-Learn is well-known for its emphasis on model interpretability and evaluation. It offers a number of tools for feature selection, cross-validation, and model metrics. With all these features, Scikit-Learn is a must-have library for machine learning professionals, regardless of experience level or volume of intricate real-world projects being worked on.

Chapter 3

Approach

In this chapter will presents the approach adopted to solve the problem introduced before, presenting the architecture that was chosen and explaining how the situation was handled.

3.1 Proposed Solution

In order to achieve the solution to the problem related previously, it is essential to define a network topology that satisfy the main reasons of using a SDN controller and a system that support the idea of scalability that can handle the large amount of data with an easy way to add more resources whenever necessary. With this idea in mind, the solution scheme will be presented in two phases, the network topology and the system architecture.

3.1.1 Network Topology

Thinking in simulate a more realistic scenario, a set of hosts will be connected to a virtual switch allowing communication between each other, and defining a collection of Opendaylight nodes as the controller of the network, as it can be see in the figure 3.1.

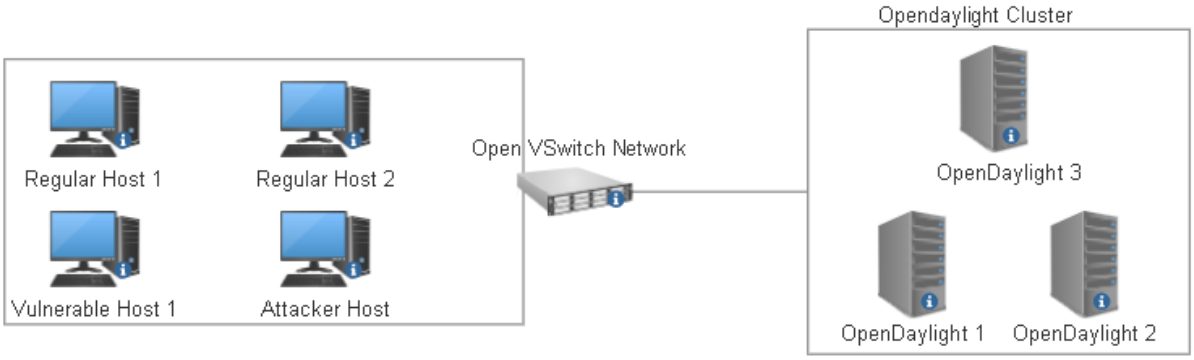


Figure 3.1: Network Topology

The hosts will assume different roles in this work, as ordinary hosts, vulnerable hosts and attackers. The Opendaylight cluster will allow a distributed deployment starting with three nodes, in order to maintain the service work in cases of attack or some failure resulting in the fall of one of this nodes. All the traffic generated in the network will be monitored, captured and transmitted to the following part of the system that gonna be presented in the next section.

3.1.2 System architecture

Searching for a system that can handle a real-time packet capture, is essential that it can provide scalable resources satisfying any condition that is needed to process, transmit, store and analyze the data during the process. As network traffic grows exponentially, a scalable solution guarantee that the system can handle increasing volumes of packets without compromising performance.

A scalable system enhances the efficiency of packet analysis and processing, ensuring timely analysis and response. This is particularly critical for cybersecurity and network monitoring, where any delays in packet processing can result in several consequences. Also it allows fault tolerance and resilience, distributing packet capture and processing across multiple nodes, ensuring redundancy and fault recovery in the event of a failure or

increased traffic load.

The main idea for this architecture can be seen in the figure 3.2 where it will permit the system to capture, transmit, store, and analyze all in a scalable manner.

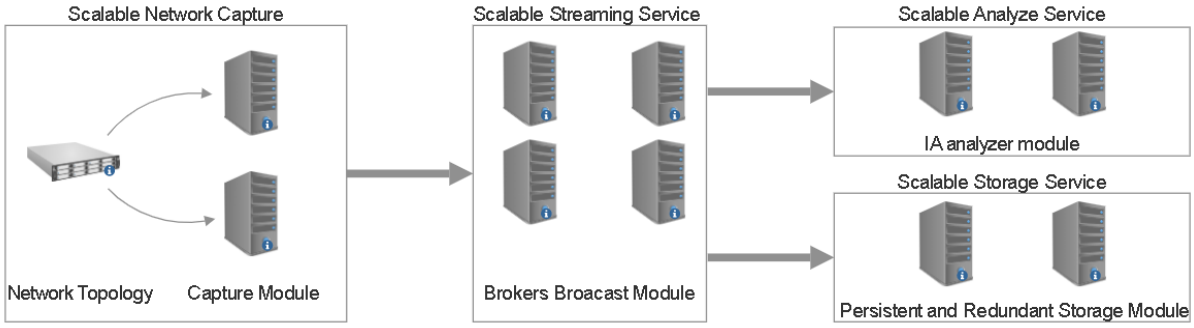


Figure 3.2: Scalable System Architecure

3.1.3 Packet capture

Analyzing the studies from the previous chapter, the mainly roads to capture the traffic from the SDN networks is forwarding the packets to a packet analyzer or mirror the traffic in the virtual switch to a new port and store it in a pcap file. Since we don't want to interfere in the flow of the network or on it's performance, we opted for using the mirroring option in the OVS.

In the documentation of the OpenVSwitch and OpenDaylight, them recommend the usage of the tool TCP Dump., which is a packet sniffing and network analysis tool used in Unix-like operating systems, allowing users to capture and analyze network traffic in real-time and exporting it through PCAP files. TCP Dump provides a wide range of options and filters to customize the packet capture process and extract specific information from network packets.

TCP Dump can be combined with other tools, in-depth packet analysis and fits perfectly to our architecture which can be used by the producer to sent the data to the external system for the further processing. It's important to note that TCP Dump operates at a low-level network layer. The perfect scenario would it be a tool or an application that could be run in a more higher level application which could be

connected to the ODL features.

When running TCP Dump, it listens to a specific network interface, capturing all incoming and outgoing network traffic on that interface which will be the SDN interface that connects to the Virtual Switch.

3.1.4 Data set Generation

To train the Artificial intelligence to recognize malicious packets, is necessary to collect these types of packets and create data sets with some types of attacks. Data sets are a crucial role in training artificial intelligence, as they serve as the foundation upon which AI models learn and make predictions or decisions. This kind of information to the AI models of examples and related points of the problem, allowing them to learn patterns, relationships, and correlations.

With the training, the analyze algorithms can identify relevant features and extract valuable information. This learning process enables the AI model to generalize knowledge and make predictions or decisions on unknown data. This enables AI systems to adapt and make informed predictions or decisions in real-world scenarios that may differ from the training data.

In order to create this data sets, we will use the attacker host to execute a battery of attacks and will store each of these generated packets in a PCAP file, which will be kept in the HDFS node for further analysis. Also thinking in cover more malicious activities, aa complementary module will be developed using a passive DNS technique.

3.1.5 Analysis module

The analysis stage follows after gathering both harmful and legitimate network traffic and involves a thorough study of the data packets that were recorded. The analysis process is diverse and includes a range of procedures and strategies to pick up on the smallest details that could distinguish malicious from authorized activity.

The core of this research is the examination and dissection of packets. The core

components of each data packet are revealed through dissection. Examining the payload, which comprises the actual data being transported between the source and destination, as well as the packet headers is part of this. Analysts can spot telltale signals of malicious intent because to the useful metadata provided by header information, which includes IP addresses, ports, protocol types and timestamps.

Behavioral analysis is essential in addition to static analysis. Analysts keep track of the behavior of network packets and flows throughout time. Unusual traffic patterns, such as excessively high data transfer rates, repeated connections to domains that have never been observed before, or patterns resembling port scanning, might all be warning signs. This method is especially useful for identifying fresh threats that might not have established signatures.

Machine learning techniques are rapidly being used into intrusion detection systems to automate and speed up this process. These algorithms develop their recognition of complex patterns linked to both benign and dangerous traffic as a result of learning from prior data. This considerably improves the system's capacity to identify previously unidentified threats by enabling real-time examination of incoming packets against a continuously updating knowledge base justifying the main idea of building an AI model to help trough this process.

3.1.6 AI Model

For the development of the AI model we will balance the data set generated with all the information needed for the AI to analyze and classify the data in a real time analysis. This approach was based on some examples that were presented before in the state of art, were the authors tests several machine learning algorithms trying to find the most accurate to their problems.

In order to evaluate the best choice for our research, four main metrics were chosen to give the best information trough the Algorithms in the test. And they are:

Accuracy

Accuracy is a fundamental metric that measures the proportion of correctly classified instances among all the instances in the data set. It is calculated as the Accuracy A given the number of true positives TP, the false positives FP, the false negative FN and the true negative TN:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Recall

Recall measures the proportion of actual positive instances that the model correctly predicted as positive. It focuses on the ability of the model to capture all relevant instances of a particular class. It is calculated as:

$$R = \frac{TP}{TP + FN} \tag{3.2}$$

High recall indicates that the model is good at identifying most of the positive instances, but it might have more false positives.

Precision

Precision measures the proportion of instances that the model correctly predicted as positive, out of all instances predicted as positive. It reflects the model's ability to make accurate positive predictions. Precision is calculated as:

3.3:

$$P = \frac{TP}{TP + FP} \tag{3.3}$$

High precision means that when the model predicts an instance as positive, it is more likely to be correct, but it may miss some positive instances (lower recall).

F1 Score

The F1 score is the harmonic mean of precision and recall and provides a balanced measure of a model's performance. It is particularly useful when there is an imbalance between the classes or when both precision and recall are crucial. The F1 score is calculated as:

$$F = \frac{2(P * R)}{P + R} \tag{3.4}$$

The F1 score balances the trade-off between precision and recall, and it is especially valuable when you want a single metric that considers both false positives and false negatives.

3.1.7 Complete Solution Model

With the approach defined and the technologies that were explained in the last chapter we reached the final mkodel that will be implemented as it show in image 3.3. Starting from a virtualized network build on the top of the OVS tool, creating an virtual switch that connect all the host and allowing Openflow Protocol. Then defining the ODL framework as the controller of the virtual network.

Aligned to capture all the traffic, the TCPDump plays a pivotal role in gathering all the inside traffic and sending it to the queuing module, which have kafka and zookeeper tool in order to receive this informations and pass it to the Storage Module for possible further monitoring and for the analysis module.

In the analysis the development of a AI model where it can check each packet information and give a prediciton if it is malicious or not.

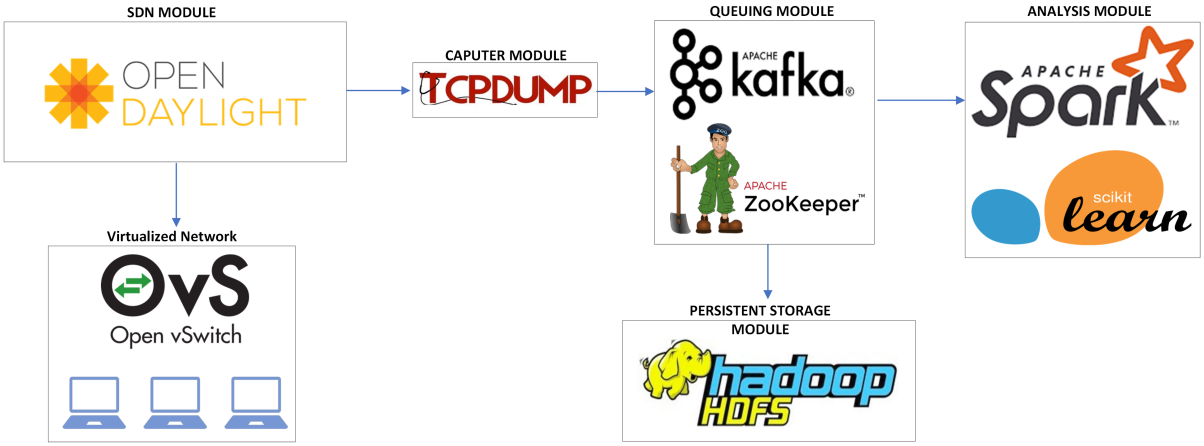


Figure 3.3: Complete Technologies Architecure

Chapter 4

Implementation

This chapter will explain the implementation of each one of the layers of the system architecture and the configuration of the network, describing the problems encountered and how they got surpassed, as well all the configuration about the system.

4.1 Network Deployment

The first part of the implementation is to deploy the network with all the hosts connected and setting the controller to the traffic. The network was divide in two, one (SDN topology figure 3.1) created by using a virtual machine inside the IPB network and another (Solution Architecture figure 3.2) using several machines also in the IPB network were all the system will be connected.

4.1.1 Controller Deploy

Since it was possible to see that the controller will have three nodes to test the distributed deployment of the Opendaylight controller, each of these nodes will have the same system specification, presented by the table 4.1.

Operating System	22.04.1-Ubuntu GNU/Linux
Storage	210 GB
Processor	AMD EPYC 7452 32-Core Processor
RAM	32 GB
IP Node 1	192.168.191.86
IP Node 2	192.168.194.4
IP Node 3	192.168.191.14

Table 4.1: Controller Host Information

The Opendaylight controller used is the version Oxygen 0.8.4, the configuration file is presented in Appendix B.2. This controller is based on API as explained before and all functions that it can execute is necessary to install some inside feature for the desired purpose, so to install and configure it, was necessary to follow these steps (Complete Script in Appendix B.1):

1. Update and Upgrade the system.
2. Opendaylight is a java based application so is necessary to install java as well.
3. Configure the Java path in the environment variable.
4. Download the controller package in the distributor website.
5. Extract the files in a directory.
6. Execute the file "karaf" inside the directory /bin of the extracted data.
7. If it's everything set, the Opendaylight terminal will open (figure 4.1)

With all this set up, it's possible to start working with the controller. The controller come with only basic functions to administrate the resources, like installing the wanted features for the desired purpose.

In this work were selected several features released by the Opendaylight to reach our goals, each of them are listed and explained ahead:

- odl-dlux-core: serves as the foundation for building the DLUX user interface, providing a framework for creating a web-based GUI that enables network administrators to visualize, manage, and monitor the SDN infrastructure. (figure 4.2)

- odl-dluxapps-nodes: enhances the DLUX user interface by focusing on network nodes' visualization, management, and monitoring. It allows network administrators to view and interact with network devices, gather information about node configurations and status, monitor node health and perform node-specific management actions.

- odl-dluxapps-topology: provides network visualization, topology monitoring and analysis capabilities. It allows network administrators to view the network's physical and logical structure, monitor real-time updates, analyze paths and receive event notifications related to the network topology. (figure 4.3)

- odl-restconf: Network administrators and developers can programmatically interact with the ODL controller, manage network configurations, retrieve operational data, receive event notifications and automate network management tasks. RESTCONF's RESTful nature and adherence to standardized data modeling making it a powerful tool for implementing SDN-based network management and control.

- odl-L2Switch: enhances the OpenDaylight controller's capabilities by providing Layer 2 switching functionalities. It allows for efficient packet forwarding, MAC address learning, loop prevention, VLAN support and integration with other ODL modules, enabling network administrators to manage and control Ethernet switches in an SDN environment. Some of the main advantages are: MAC address learning and association, Packet forwarding, Broadcast and multicast handling, VLAN support.

- odl-mdsal-clustering: ODL-MD-SAL-Clustering provides a robust and scalable solution for building highly available and reliable SDN deployments. It leverages

clustering techniques to distribute the workload, improve performance, and ensure fault tolerance, making it suitable for enterprise networks, data centers, and service provider environments.

All these features were essential to the development of this work, each one of them since you install them in the Opendaylight terminal they will start working after a few minutes. The only feature that need some configuration is the MD-SAL clustering, where is needed to rewrite the IP addresses of the others nodes of the cluster in each of the Opendaylight machines.

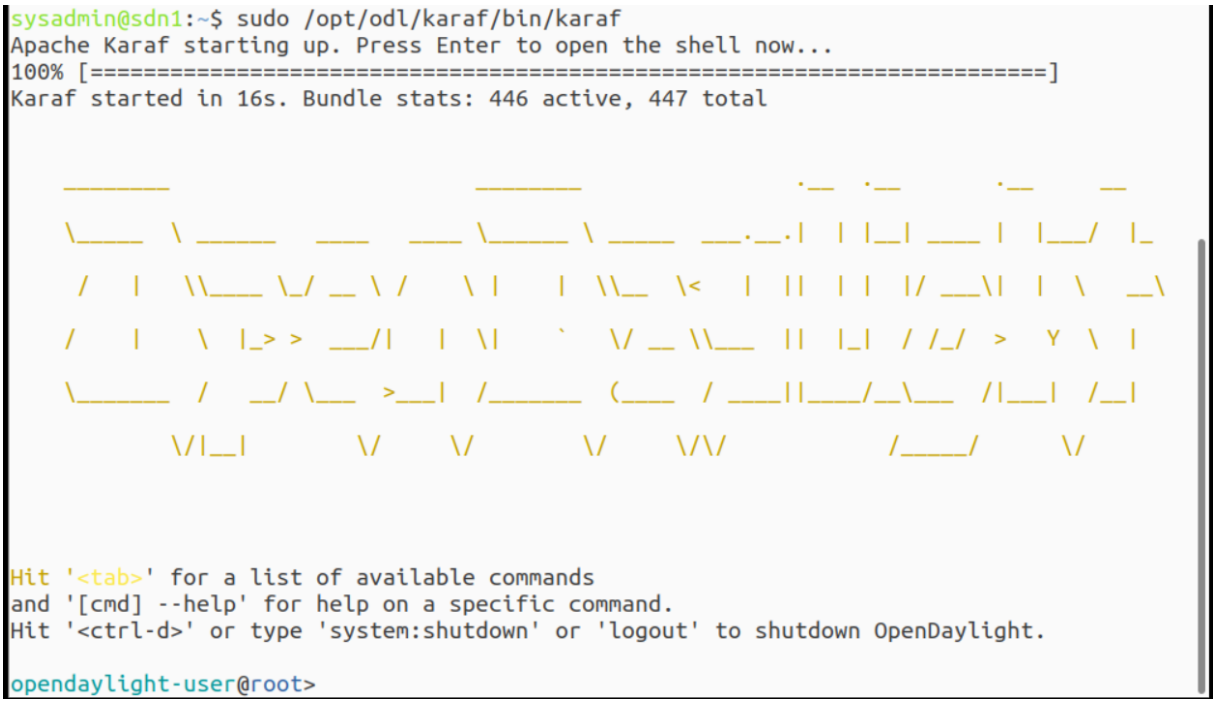


Figure 4.1: Opendaylight terminal

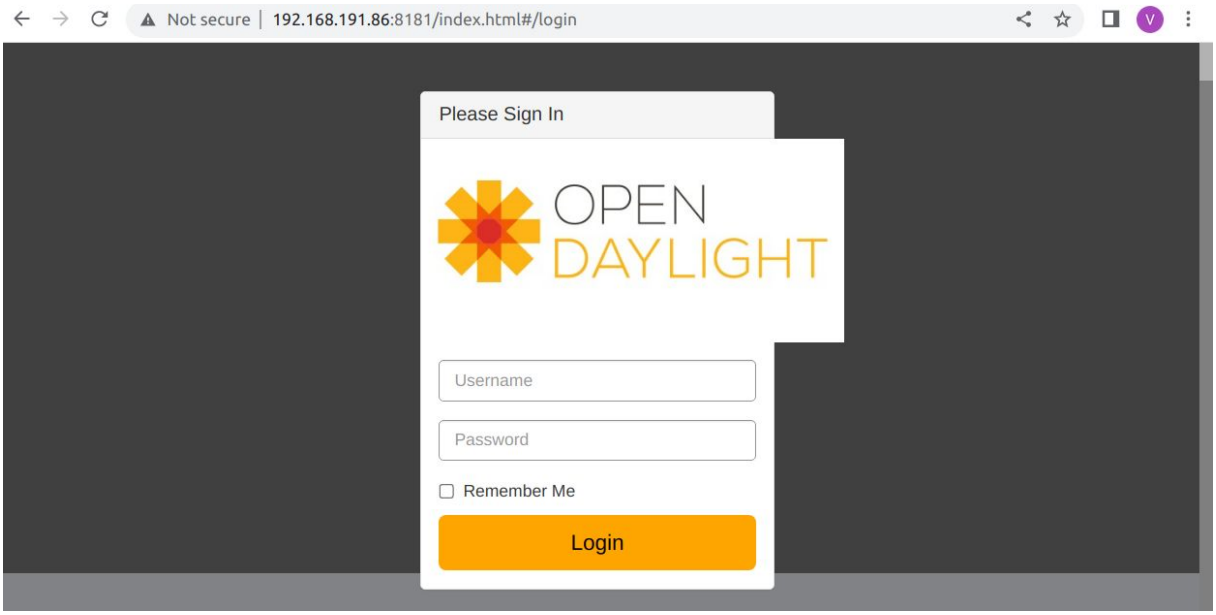


Figure 4.2: Opendaylight Dlux interface

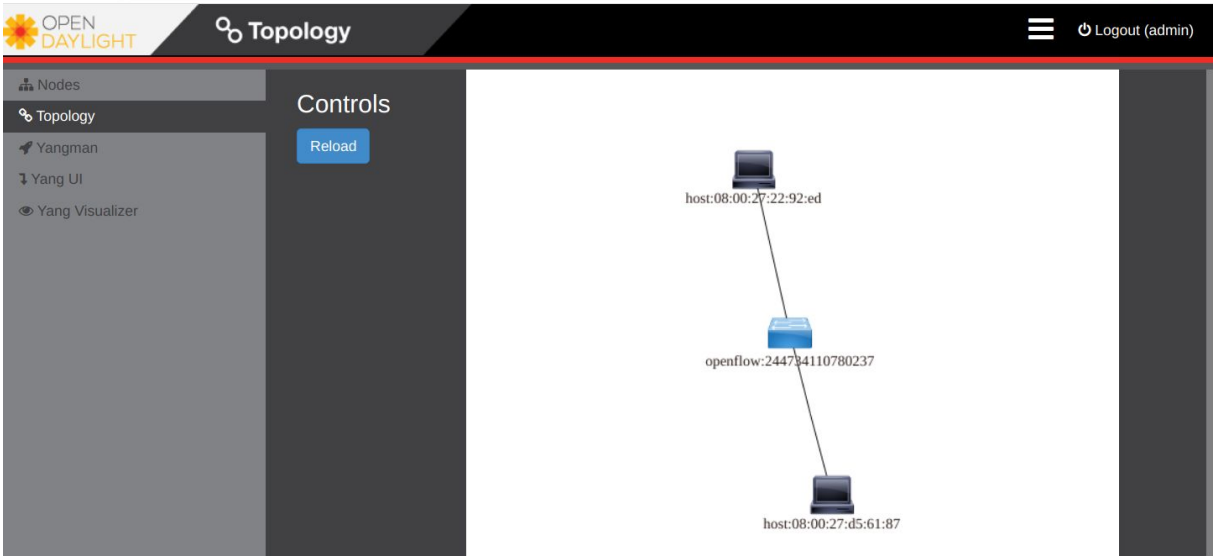


Figure 4.3: Dlux Topology feature

4.1.2 Virtual Switch Deploy

To switching service, it's used the OpenVSwitch version 2.17.5. To use and configure the service is need to follow some steps (Appendix C.1):

1. Update and Upgrade the system.

2. Install the service

3. Check if the system is running executing "sudo ovs-vsctl show"

4. Create the bridge where the hosts will connect

5. Define the protocol version that you desire (OpenFlow 1.3 used in this work)

6. Turn the bridge interface to UP state and assign the network ip address that the OVS will work on (figure 4.4)

```
sdn-br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.1.1.1  netmask 255.255.255.0  broadcast 20.1.1.255
        inet6 fe80::dc95:99ff:febc:cb4d  prefixlen 64  scopeid 0x20<link>
        ether de:95:99:bc:cb:4d  txqueuelen 1000  (Ethernet)
        RX packets 134465  bytes 14692797 (14.6 MB)
        RX errors 0  dropped 129616  overruns 0  frame 0
        TX packets 6592  bytes 1372930 (1.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4.4: Bridge interface

After this procedure the service is ready to start to be used. To connect the hosts in the OpenVSwitch network, is necessary to create a Tun/Tap device to each of the hosts that are going to be connected to the switch, which is a virtual network device that allows user-space programs to interact with network traffic at the network layer (Example of interfaces in Appendix D.1). The main steps to connect a host to the switch are:

1. If it is a virtual machine create a Tuntap device.

2. Add the interface in the Open Vswitch through "sudo ovs-vsctl add-port bridge tuntap_device".

3. After adding all the host, it's necessary to set the controller IP with the command "sudo ovs-vsctl set controller bridge_name TCP :ip_of_controller:controller_port".

4. Check if the device and the controller were added in the bridge: "sudo ovs-vsctl show" (figure 4.5 or Appendix C.2).

```
sysadmin@sdn1:~$ sudo ovs-vsctl show
[sudo] password for sysadmin:
4e0985ca-d77c-4f2f-b610-cd881464081d
    Bridge sdn-br
        Controller "tcp:192.168.191.86:6633"
            is_connected: true
        Port sdn-br
            Interface sdn-br
                type: internal
        Port host1
            Interface host1
        Port host2
            Interface host2
    ovs_version: "2.17.5"
```

Figure 4.5: Bridge with devices

4.1.3 Host Deploy

In order to deploy the hosts in a way that we can simulate attacks and operate them to execute some tests, we opted to use virtual machines to deploy this hosts, were there going to be three types of hosts: Vulnerable machine, Attacker machine, and the common hosts.

When starting the VMs is necessary to attribute their network device to the Tun/tap port created in the previous section of the OpenVSwitch which will allow the hosts communicate to each other inside the OpenVSwitch network. Also since OpenVSwitch doesn't work with a DHCP as explained in the basics configuration of the documentation[36], is necessary to configure the IP address of all hosts inside itself, which can be done using the commands with root privileges:

- ip addr add "ip_address/netmask" broadcast "broadcast_address" dev "net_interface"

- ifconfig "net_interface" "ip_address" netmask "netmask_address" broadcast "broadcast_address"

With the IP addresses set, to enable the communication now is just necessary to define the OpenVSwitch bridge as the default gateway of the packets sent by the host. This can be achieved using the following command with root privileges:

- route add default gw "ip_of_gateway"

The Vulnerable machine will have the system specification presented by table 4.2. where it will run a metasploitable 2 image which is preloaded with a variety of intentionally vulnerable software, including outdated and insecure versions of web servers (such as Apache and Tomcat), database servers (such as MySQL and PostgreSQL), and other network services (such as FTP and Telnet). Additionally, common vulnerabilities and misconfigurations are present in the system, making it an ideal target for testing various exploitation techniques.

Operating System	Linux metasploitable 2.6.24-16-server i686 GNU/Linux
Storage	16 GB
Processor	1 core gpu
RAM	4 GB
IP	20.1.1.2

Table 4.2: Vulnerable Host Information

The attacker host, will count with a Kali linux image[37], running in the system specification presented by the table 4.3. with versatile operating system specifically designed for penetration testing, ethical hacking, and cybersecurity professionals. It is an open-source distribution built upon the Debian Linux distribution and provides a wide range of tools and utilities for various security-related tasks, that will fit in the tests and to build our data set with these attacks.

Operating System	Linux kali 6.1.0-kali5-amd64 x86 64 GNU/Linux
Storage	90 GB
Processor	1 core gpu
RAM	8 GB
IP	20.1.1.3

Table 4.3: Attack Host Information

The common hosts will run a simple linux server image, and have the system specification in the table 4.4. they will have the role of generate normal traffic, to execute performance tests in the SDN network and in the future to test if the AI can differ their normal traffic to the malicious traffic.

Operating System	Ubuntu 22.04.2-live-server-amd64 x86 64 GNU/Linux
Storage	12 GB
Processor	1 core gpu
RAM	2 GB
IP	20.1.1.4 & 20.1.1.5

Table 4.4: Regular Host Information

4.2 Queuing System

Apache Kafka and the Apache Zookeeper tool were used for the queuing system of this work. Zookeeper is necessary to execute the Kafka broker leadership election and the topics partition, as well as to maintain the state of all nodes in the cluster. Both of them have a main role to execute in the queuing method.

Kafka assumes the distributed messaging system that provides a platform for building real-time streaming applications, handling the high volumes of data streams in a fault-tolerant and scalable manner. Kafka allows producers to publish messages to topics, and consumers can subscribe to those topics to receive and process the messages as explained

in previous chapters.

Zookeeper serves as a distributed coordination service that provides a centralized infrastructure for maintaining configuration information, naming, synchronization, and group services in distributed systems. It ensures that distributed processes or components can coordinate and synchronize their activities effectively.

Kafka uses ZooKeeper for cluster membership and leader election where keeps track of the Kafka broker nodes (servers) and their status. It helps in determining which broker acts as the leader for a given partition and handles leader election in case of failures or changes in the cluster. Also is used to store and manage metadata related to topics, such as the number of partitions, replication factor, and the mapping of partitions to brokers. This information is crucial for proper data distribution and fault tolerance within the Kafka cluster. The four nodes of this services utilizes the system specification defined in the table 4.5.

To configure and execute this services is necessary to follow these steps:

1. Update and Upgrade the system

2. Install the java

3. configure the SSL certificates to have access through the services

4. Download and install the service

5. configure the operational part of the Kafka/zookeeper

6. Start the service

Operating System	Debian 5.10.136-1 x86_64 GNU/Linux
Storage	150 GB
Processor	KVM 4-Core Processor
RAM	16 GB
IP Node 1	192.168.44.101
IP Node 2	192.168.44.102
IP Node 3	192.168.44.103
IP Node 4	192.168.44.104

Table 4.5: Kafka Host Information

4.2.1 Data distributing and message sizes

Messages in Kafka are ordered per partition. When consuming from a topic with multiple partitions, the order of retrieved messages is not guaranteed to match the publishing order, bringing a serious problem when we talk about network packets, since they need to be together in order to have a trustful and complete data. Having only one partition ensures message order, but it fails in the mission of bringing scalability to this work, so to fix this problem was used a strategy of resizing the message size to don't leave any packet information in different messages.

When talking about how Kafka works in different partitions and message sizes, it's necessary to test and compare the performance of each characteristics to combine the best options, bring the best performance that it can take. This comparisons are gonna be made in the following chapter of results and discussion.

4.3 Storage System

To storage all the packet files received by the Kafka, the HDFS storage was chosen using the version 3.2.2 of the Apache Hadoop. To install and execute it's services were used similar steps like the other tools:

- Update and upgrade the system.
- Install and configure java environment variables.
- Download and install the Hadoop tool.
- Configure the services and it's nodes.
- Configure the access through the SSL certificates.
- Start the services.

No bigger problems or variations were found in this part of the implementation. The specification of the machines where the nodes were installed are defined in the following table 4.6:

Operating System	Debian 5.10.136-1 x86_64 GNU/Linux
Storage	150 GB
Processor	KVM 4-Core Processor
RAM	8 GB
IP Node 1	192.168.44.105
IP Node 2	192.168.44.106
IP Node 3	192.168.44.107
IP Node 4	192.168.44.108

Table 4.6: HDFS Host Information

4.4 Capture system

Analyzing the studies made by other authors in the chapter 2, was possible to see that when talking about the security of SDN, the authors uses forwarding rules to redirect the traffic coming to the controller to a external tool like Packetfence or Snort. This kind of approach interferes directly in the traffic performance of the network, and the objective of this work is to mirror this traffic in order to don't affect it's performance.

Looking for some techniques or approaches used to solve this problem, nothing that was implemented attend to this criteria, same implementations using forwarding rules to external tool. Beside one only experiment found ten years ago, implemented in java language that could replicate and capture this traffic in a maven application. But it was designed for libraries and components that ended through the time.

So looking for solution in the switching layer, was found by documentations of the controller and other experiments that TCP Dump python library could handle this problem, by capturing the Openflow traffic that comes to Opendaylight.

To capture the traffic we used the python language, which has large amount of information and libraries that can contribute to the progress of this work. These libraries are:

- confluent_Kafka; A feature-rich and high-performance Python library for interacting with Apache Kafka. It provides a comprehensive set of functionality for producing and consuming messages, supporting schema registry integration, delivery reports, partition management, consumer groups, security features, and extensive configuration options(Appendix E.2).

- hdfs: HDFS library in Python enables developers with the Hadoop System using the Python programming language. It provides functionalities for file operations, data streaming, configuration, authentication, error handling, and integration with the Hadoop ecosystem.

- subprocess: This library has a flexible interface for managing sub processes and executing external commands. It allows you to automate tasks, run shell commands, capture output, and interact with external programs seamlessly from within your Python scripts. This library is essential to execute the TCP Dump process that is installed inside the machine, and to capture the response.

With these libraries it's possible to execute a code where it open a process for the TCP Dump tool to capture the traffic that reaches the controller. To differ the traffic of the

controller with the normal traffic of the computer, we capture only the packets that goes to the port of the controller (6633). After capturing the traffic, all the packets are wrote in a PCAP file, which will be published in the Kafka with the confluent Kafka library. We establish a connection through SSL certificates, decoding and sending the data of the PCAP, distributing the information among the nodes. In order to solve the problem of network packets been cut by the size of the Kafka messages, the script get the information from the packet header of it's size and if is lower or equal of the size remaining in the message, it will fit in and send. Besides if is bigger than the space remaining, it will send the message with the current content and start a new message with this packet. All this process can be easy seen in the next figure 4.6 and the code are presented in Appendix E.3.

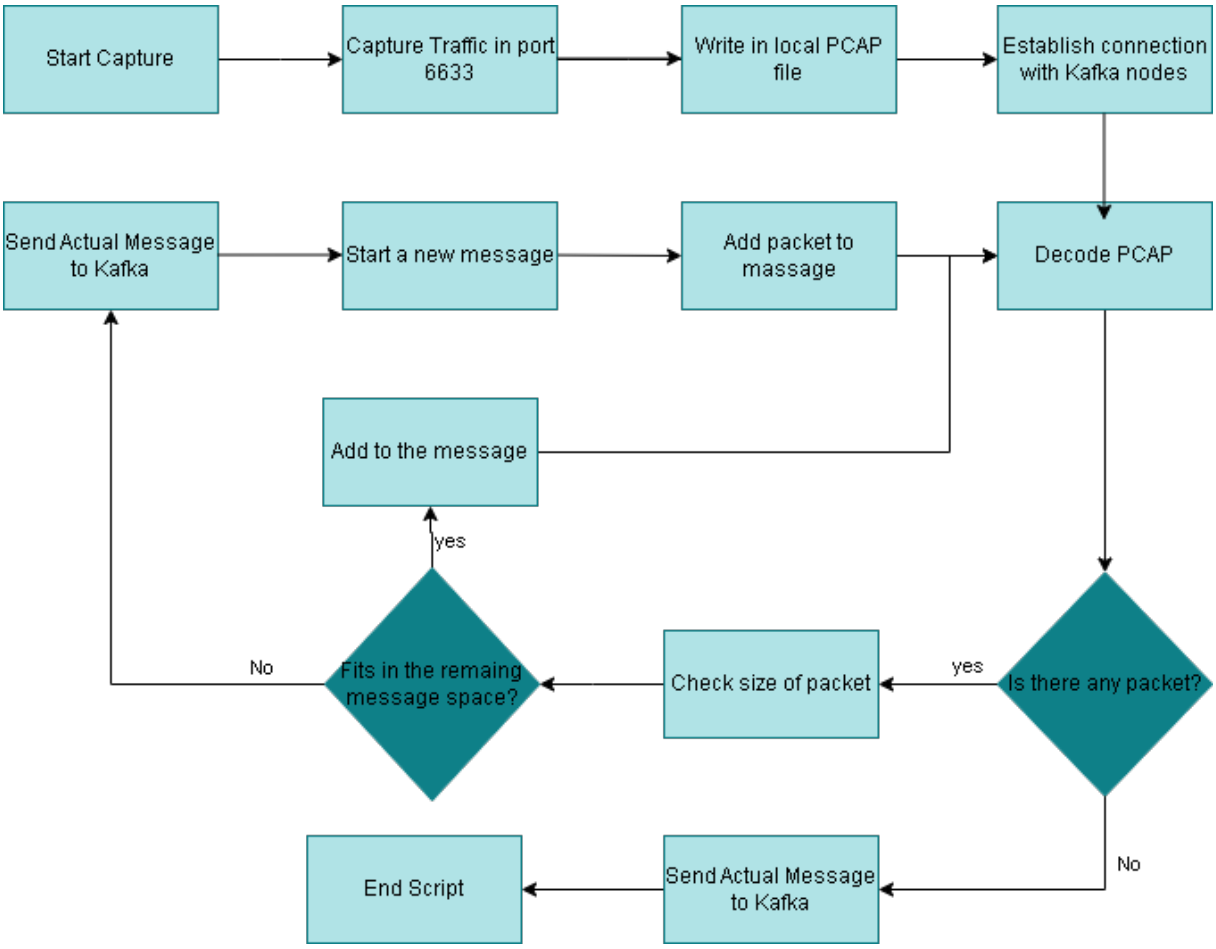


Figure 4.6: Capture Fluxogram

4.5 Dataset Collection

After all the environment set, it's possible to start the attack phase, where all the attack vectors that are going to be tested are executed by the attacker host focused in the metasploitable host. Each of these attacks are going to be captured by the capture system, and sent for the persistent storage (HDFS) labeled by the tool and categorized.

The attack vectors that are going to be tested are listed in the table bellow, each of them were tested and captured part of each other, searching to identify and isolate the packets of each attack. The results of these collection can be seen in the next chapter.

Attack Vector	**Description**	**Tools**
Network vulnerabilities mapping	Network mapping's objective is to collect information about the network's assets, such as IP addresses, open ports, operating systems, services running on hosts and network protocols in use. This data assists administrators and attackers in understanding the network's structure, aiming to discover weaknesses that could be exploited by attackers to compromise the target's confidentiality, integrity or availability.	OpenVAS
Port Scan	A port scan is a method for locating open ports on a target machine or network. Sending network packets to various ports on the target and evaluating the answers to identify whether ports are open, closed, or filtered is the process.	Nmap

Continued on next page

Table 4.6: (Continued)

DDoS	A DDOS attack is a malicious effort to interrupt the regular operation of a network, service, or website by flooding it with unauthorized traffic. The assault is called distributed because it often involves numerous hacked devices working together to establish a botnet in order to produce excessive bandwidth.	Hping3
Brute force	Brute force attacks in Telnet involve systematically attempting multiple combinations of usernames and passwords to gain unauthorized access to a remote Telnet server. Telnet is a protocol that allows users to remotely access and manage devices or systems over a network.	Telnet
Misconfigurations	Misconfiguration and weak password attacks are common techniques used by attackers to exploit vulnerabilities in systems and gain unauthorized access.	CVEs

4.6 Analysis module implementation

We methodically divide the development process for the analysis module into separate phases, each of which contributes to the establishment of a sophisticated AI model capable of analyzing collected network traffic in the SDN environment. Beginning with data collection and preparation, where the raw network packets are ingested and converted into a suitable format for subsequent analysis, this complex method entails a number of clearly defined processes. The module then starts the feature extraction and engineering process, obtaining important properties and parameters from the packets that help determine

whether they are malicious or benign. The process will be presented and each of the choices are going to be used in the end of the work.

4.6.1 Traffic Analysis

Starting thinking about the traffic analysis, the first main step is to be capable of braking the packets in the parts that are important for the process. Such as the header information and the body of the massage inside the packet.

Working with the PyPacker library it is possible to access all the package information gathered inside the PCAP file. As it can be see in the code presented in the appendix E.6 we run through all the packets, collecting and storing the information that we judged necessary to identify the possibility of malicious activity.

Trying to classify and select the most important fields to train our model, we reached in these list of information that can be extracted from the packets, based on the type of attacks that we executed and explained in the data set collection.

- Internet Header Length: The IHL has four bits that specify the number of 32-bit words in the header
- Service Type: This field provides the queuing of the IP packets in their transmission
- Total Length: This is the total size of the header and data in bytes, where the minimum size of the Total Length field is 20 bytes and the maximum size is 65,535 bytes
- Identification: If the IP datagram is fragmented (broken into smaller pieces), the ID field helps identify fragments and determine to which IP packet they belong to
- IP Flags: This is a 3-bit field that uses a few possible configuration combinations of control flags for fragmentation
- Fragmentation Offset (Fragment Offset): The Fragment Offset field, which occupies 13 bits, performs packet tracing by indicating the data bytes ahead. It does this by

identifying where in the original packet a specific fragment belongs.

- Time to Live (TTL): TTL restricts the datagram's lifetime by forcing undeliverable datagrams to be automatically deleted, preventing packets from an unending cycle in the internet system.
- Protocol: This 8-bit field specifies the protocol used in the packet's data section.
- Header Checksum: The Header Checksum field recognizes any communication faults in the header.
- Source IP Address: IPv4 address of the sender of the packet
- Destination IP Address: IPv4 address of the receiver of the packet
- Source port: Specifies the port number of the sender.
- Destination port: Specifies the port number of the receiver.
- Sequence number: During a TCP session, how much data is sent is indicated by the sequence number. Utilizing this sequence number, the recipient replies with an acknowledgement.
- Acknowledgment number: Used by the receiver to request the next TCP segment.
- TCP Flags: Flags take up 9 bits, also known as control bits, are employed to create connections, transmit data, and break connections.
- Window: The receiver's acceptable number of bytes is specified in the field. In order to inform the sender that it would like to receive more data than it is currently receiving, it is used by the receiver.
- TCP Checksum: Used for a checksum to check if the TCP header is OK or not.
- Urgent Pointer: The urgent pointer serves as a marker for the end of the urgent data when the URG bit is set.

- Body: Where all the data thar are willed to be received by the receiver are stored.

With this collection of data, we can also calculate and identify some important information, such as the amount of packets that are been exchange between hosts, the ports that are been loaded with information from the sender, the validation of the data inside the packet. Now with this field is possible to train the AI model.

4.6.2 Passive DNS Implementation

As mentioned before the passive DNS solution are implemented based on library developed by the CIRCL Passive DNS database, for querying Passive DNS records, extracting relevant data, and preparing it for analysis.

The data for each domain that will be searched on are presented as a string, encapsulating multiple records like:

- Rrname: DNS record name
- Rrtype: Type of DNS record
- Rdata: Data associated with the DNS record
- Time_first: Timestamp of the first occurrence of the DNS
- Time_last: Timestamp of the last occurrence of the DNS

with all the information that we can catch from requesting to the CIRCL database, we can perform some operation in order to try to identify the legitimacy of these Domains. Based on a combination of approaches explored by Khalil, Yu, Guan[38]. and Liu, Zeng, Zhang, Xue, Zhang, [39]. We reached in some important lexical and resolving features of the DNS such as:

- Number of distinct type 'A' records
- Number of distinct 'NS' records

- IP entropy of domain name
- Similarity of NS domain name
- Max length of labels in sub-domain
- Length of domain name
- Character entropy in DNS name
- Number of numerical characters
- Max length of continuous consonants
- Max length of continuous same alphabetic characters
- Max length of continuous numerical characters
- Max length of continuous alphabetic characters
- Ratio of numerical characters
- Ratio of vowels

By identifying and calculating these characteristics of the DNS we can manner a way of analyze the legitimacy of these servers, and train an model capable of calculating and interpret these information by extracting the DNS packets coming from the capture traffic.

In appendix E.6 it's possible to see the code developed in these step. The code consist in some steps:

- Extract DNS packets from upcoming traffic that were captured
- Extract the Domain Name Server information that are relevant for the DNS consult
- Get information about each of the DNS in the CIRCL database
- Calculate the previous characteristics from each unique DNS
- Analyze with the model developed for the DNS Analyze

4.6.3 Machine Learning Algorithm Tests

After the development on how to extract the important information and create a dataset with examples of what are malicious or not in the common traffic or in SDN, it is necessary to test a variety of machine learning algorithms leading us to a result that can effective classify the captured information in benign or not.

For this part we count wuth the scikit-learn library where we could find a great pack of algorithms to test their performance, the metrics defined to evaluate these approaches of machine learning were presented in the previous chapther, the Accuracy, Recall, Precision, f1-Score.

Taking in consideration each field, and data explained in the development of the traffic analysis and in the Passive DNS, we obtained the results in the AI tests as presented by the table 4.7 and table 4.8.

Machine Learning Algorithm	**Accuracy**	**Recall**	**Precision**	**f1-Score**
AdaBoost	0.949809	0.949893	0.949893	0.946893
Decision Tree	0.939809	0.929421	0.914853	0.929893
Gaussian Process	0.939231	0.949893	0.924345	0.939427
Linear SVM	0.919619	0.909423	0.912643	0.909872
Naive Bayes	0.933903	0.953557	0.939571	0.936580
Nearest Neighbors	0.959524	0.959893	0.929454	0.919732
Neural Net	0.951047	0.969574	0.949361	0.929467
RBF SVM	0.919619	0.909893	0.913680	0.929787
Random Forest	0.969809	0.944493	0.979893	0.967133

Table 4.7: Traffic Analysis Machine Learning Result

Machine Learning Algorithm	**Accuracy**	**Recall**	**Precision**	**f1-Score**
AdaBoost	0.937610	0.899375	0.881552	0.925617
Decision Tree	0.893141	0.878234	0.875853	0.876753
Gaussian Process	0.939642	0.909054	0.874004	0.891524
Linear SVM	0.935195	0.914256	0.931611	0.923415
Naive Bayes	0.891376	0.854155	0.877088	0.853652
Nearest Neighbors	0.939259	0.905183	0.876643	0.894211
Neural Net	0.941724	0.909830	0.920884	0.937223
RBF SVM	0.935195	0.928686	0.931611	0.915929
Random Forest	0.945462	0.929115	0.920910	0.941583

Table 4.8: Passive DNS Machine Learning Result

Analysing the results, we could conclude that the most eficient algorithm based on it's performance is the Random Forest Algorithm for the Traffic Analysis model with around 96% of accuracy in identifying the malicious packets, and in the Passive DNS approach although the Random forest have the highest accuracy, the RBF SVM algorithm show better results in a wide overview, keeping more consistent results in the metrics.

Chapter 5

Experiments and Discussion

This chapter brings the results of the experiments performed, explaining why and how it was conducted, discussing the results and the reasons of why that occurs. These experiments brings a performance test, where we wanted to know the best configuration and distribution of the queuing module (Kafka message size and partitions), the throughput of the communication between the nodes in a SDN network, the performance of the packet capture module, the AI model performance in a stress test and in a regular situation and a example of how the system worked in it's final form.

5.1 Tool used in the experiment

Since this work is focused on performing a SDN network inside the most real situation that is possible, we consider that the maximum network bandwidth that real devices like computers with ethernet cables could handle, working around 1GB/s.

To simulate this kind of network traffic, to test all the experiments that were presented previously was chosen the Iperf3 tool. An open-source application used as a network testing tool designed to measure and analyze network performance. It works in a client-server architecture where two hosts connect between themselves and transmit traffic based on bandwidth and time that the user desires. It fits perfectly to simulate our conditions and evaluate throughput, packet loss and latency.

5.2 Network Throughput

Using the iperf3 tool was possible to simulate the data transfer between hosts in order to evaluate and analyze how a SDN network act in a real scenario. To bring a comparison it was tested the throughput using the same conditions with and without an SDN controller managing the network. Simulating the bandwidth of 1GB/s during the communication between the hosts, the standard network could maintain a performance of 998 Mbits/sec, while in the other side the SDN network could perform in 972 Mbits/sec.

The Throughput collected in the experiments can be seen in the figure 5.1. Even performing in a very similar average bit-rate of the standard network, is possible to see the inconstancy of the performance during all the test. This downgrade and variation can be explained by the fact that the process of receiving the traffic on the OpenVswitch and forwarding it to the controller for the switching decision process can cost in the performance on delivering a quality and constant server, but even with this negative side, the values where close to the common network.

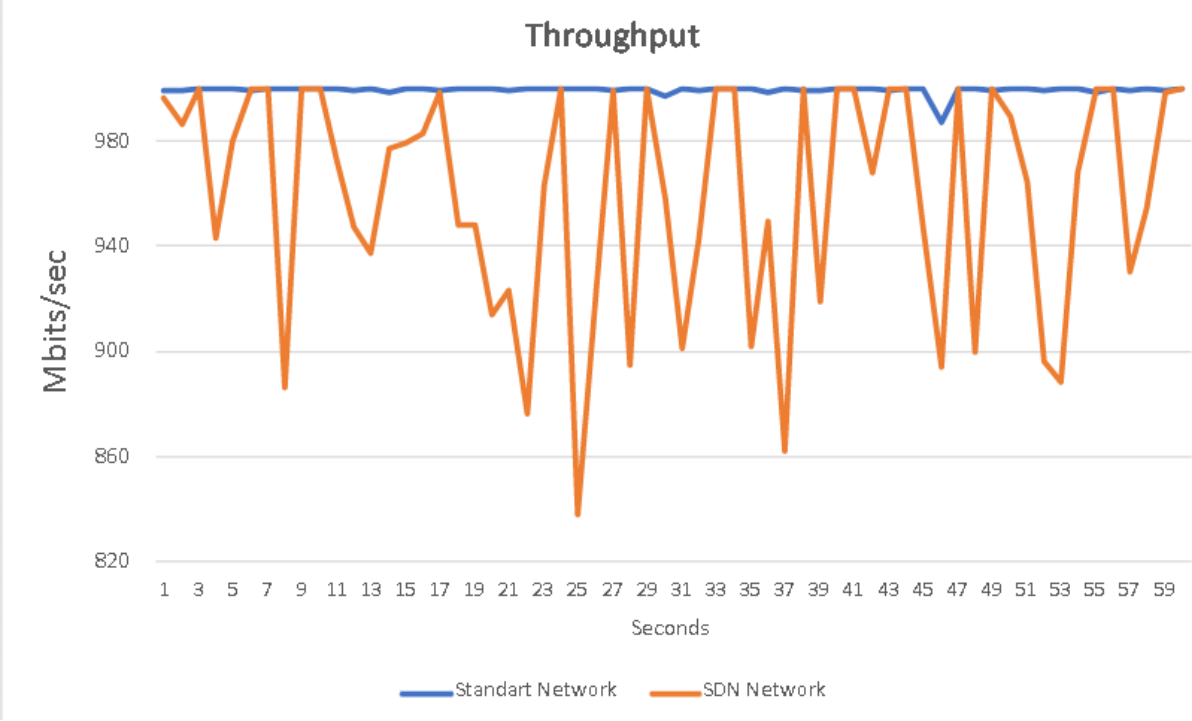


Figure 5.1: Network Throughput

	Number	Packets Received	Packets Captured	Packets Dropped	Packets Loss (%)
Tests	1	2213793	2213793	0	0
	2	4654748	4654583	165	0,003
	3	4416509	4416393	116	0,002
	4	4583945	4583710	235	0,005
	5	4685000	4682572	2428	0,05

Table 5.1: Packets Capture Result

5.3 Capture module performance

To validate if the capture approach can fit and handle the objective of this work in capturing the maximum amount of packets and don't lose a significant quantity of this information, is important to evaluate the capacity of receive, capture and send information through the system.

Simulating the same scenario of the previous experiment, using the Iperf3 to perform the packet exchange between the hosts in the maximum amount of bandwidth that it can takes, the capture module worked in a period of 60 seconds, the same of the throughput test, registering the amount of packets received, captured, and dropped before the transmission to the Kafka start.

The results are presented in the table 5.1. It is possible to see that even with the large amount of the data received, the module performed really well, following a pattern of losing more packets when the amount of packets grow, but even with this lose, the quantity of these leaks is minimal.

With the results, the approach used in this work is validated, were in normal traffic conditions (in a daily situation of a network) this amount of packets tends to be lower than the amount registered in the experiment, and in the vulnerabilities/attacks test, it will fits to the objective of capturing all these malicious packets to generate our data set.

5.4 Kafka Performance

Since the Kafka service is a scalable tool that allows you execute different configurations, it is necessary to test and evaluate the performance of each features seeking the best configuration. When we talk about security in the network, every second counts, so every gain that we can achieve by changing the configurations is important. The most important features that Kafka allows us to set is the number of partitions and the size of the messages that it can receive.

5.4.1 Kafka Partitions

The partitions are essential in scalability and redundancy in the Kafka work, where each topic you create in the service will already have 1 partition where it will store the data sequentially, but these partitions can be defined in the moment that you create a topic, and it will spread the data received by the messages among the partitions distributing on each server. With this distributing with the same number of partitions in each server, it is possible to replicate the data and guarantee the fault tolerance.

So searching for the best amount of partitions to the work, were tested a range of 1 to 4 partitions. Thhis number comes from a suggestion recommended by the documentation of the Kafka manual, where if we are working with a 1000 Mb/sec we need to divide by the amount desired by the capture module, reaching an amount of 1 to 4 partitions.

The result of this test can be seen in the figure 5.2, where when we increase the number of partitions, the delay of the process in transmitting the information to the Kafka decrease on a significant amount. Using the default configuration of a topic, the delay reached in an average rate of 213 seconds, and when we use the maximum amount that was defined, the performance increased to a range of average rate 3,02 seconds, been the best choice to this work.

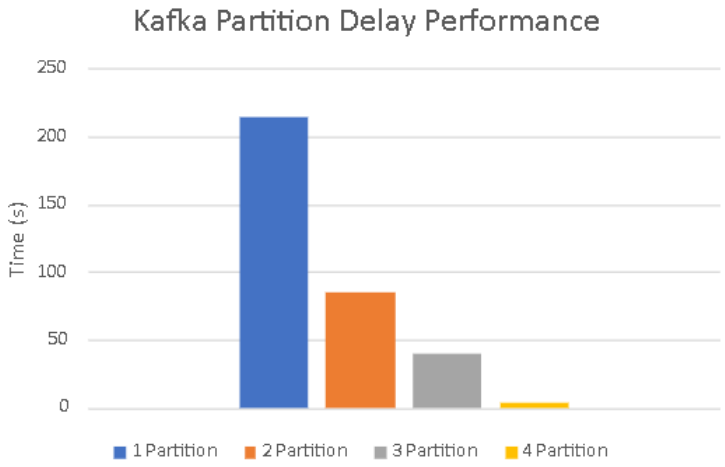


Figure 5.2: Kafka partitions performance

5.4.2 Kafka Messages

The size of Kafka messages has a direct impact upon the amount of network bandwidth necessary for data transmission. Kafka doesn't support large data to be transmitted in on a one way to the service. Larger message sizes need more bandwidth, resulting in greater network traffic. According to the confluent Kafka manual, the maximum size supported is 1 MB.

Using the capture module to test the message sizes, we can see that we are handling gigabytes of files because of the enormous amount of network packets received by the capture module. It will be necessary to test different sizes of messages, using the max size that can bring the best performance of the transmission to the Kafka module. The tested sizes were:

- 128 KB
- 256 KB
- 512 KB
- 1024 KB

The result of this test can be seen in the figure 5.3. Already using the best partitions division presented in the previous subsection 5.4.2 (4 partitions), we can see that the time influences directly on the performance of the transmission to Kafka server. Where greater the size, the speed of transmission increase as well.

It's possible to see that when increasing the size of this message the performance of the 4 partitions configuration increased in more the 2,5 seconds less than when using 128KB. Even the messages been smaller it turns necessary to send more messages consuming bandwidth of the network which can cause this performance decrease with smaller sizes.

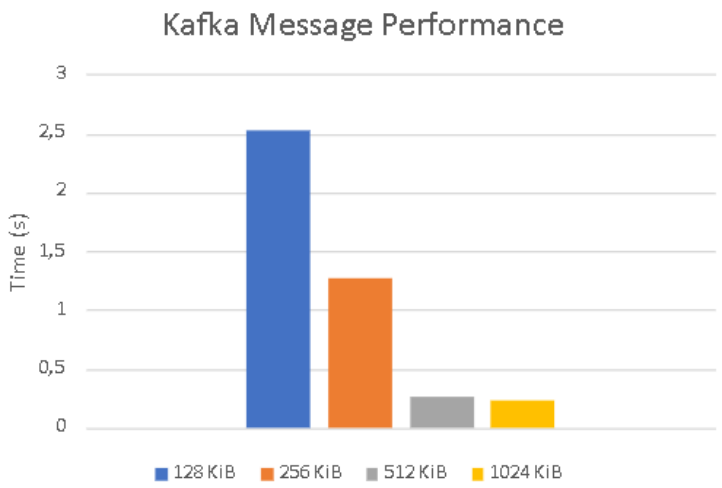


Figure 5.3: Kafka message size performance

5.5 Data Set Results

On the production of the data set related with the attacks executed in this work, we reached the number of 16.687.656 total packets (table 5.2), which provided for us consistent and balanced data set for the training model.

Type of Packet	**Total**
Benign Packets	4.895.885
Network Vulnerability Mapping	1.682.543
Port Scan	741.083
DDoS	9.340.748
Brute Force	25.152
Misconfigurations	2.245

Table 5.2: Dataset packets result

5.6 AI analysis

As presented in the subsection 4.6.3, the usage of the Random forest Algorithm for the traffic analysis and the RBF SVM Algorithm for the Passive DNS analysis, showed a good performance on identifying the malicious activities in the reaching respectively 96% and 94%.

But for concluding if this approach is feasible in a real environment, it is necessary to validate the time that the analysis takes to realize all the process. In order to identify the time spent by the code, we divided the timing in the 3 main process of the code:

- Consume of the Queuing module
- Processing the packets for the Traffic Analysis
- Processing the packets for the DNS Analysis
- Execute Traffic Analysis Model
- Execute Passive DNS Analysis Model

In the image 5.4, we can see the amount of time that all the process is gonna take with worst case with more than 1 million packets and around 500 DNS packets with a

fast consumption around 1.30 seconds, to parse all the captured packets took a time of 36 seconds to get inside of each packets and extracting all the information needed in the traffic analysis, now in consulting the passive DNS database of CIRCL to get the information needed for the analysis it took around 194 seconds. When executing the AI model responsible to analyze the traffic it run is a period of 2.71 seconds, now during the analysis of the Passive DNS analyze it reached a execution time of 9.07 seconds. Reaching a total of 246 second which is not a good result for an IDS system.

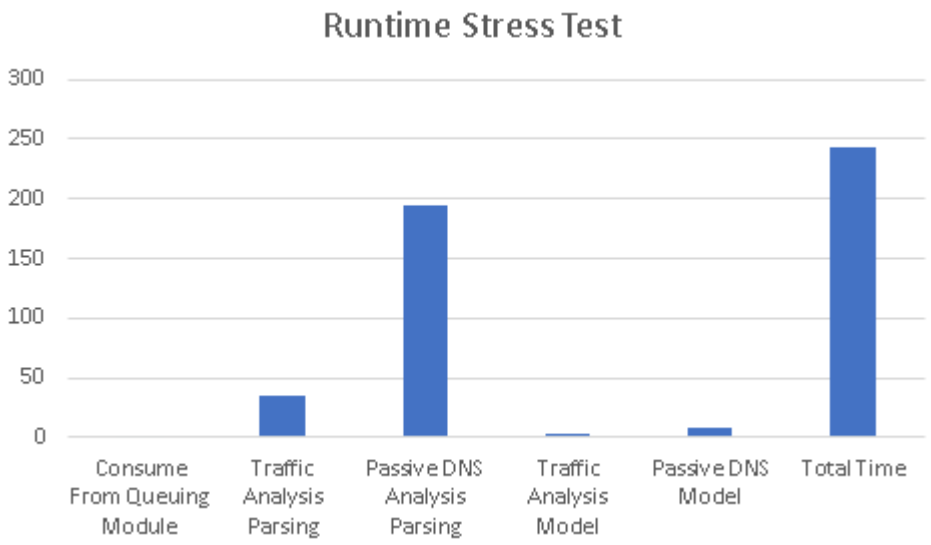


Figure 5.4: Runtime Stress Performance

If we consider a more common scenario with distinct hosts exchanging an amount of 10.000 packets which 120 are DNS packets we would reach a performance around 93 second (figure 5.5), while maintains a fast performance on the general traffic but with a slow processing in the Passive DNS solution.

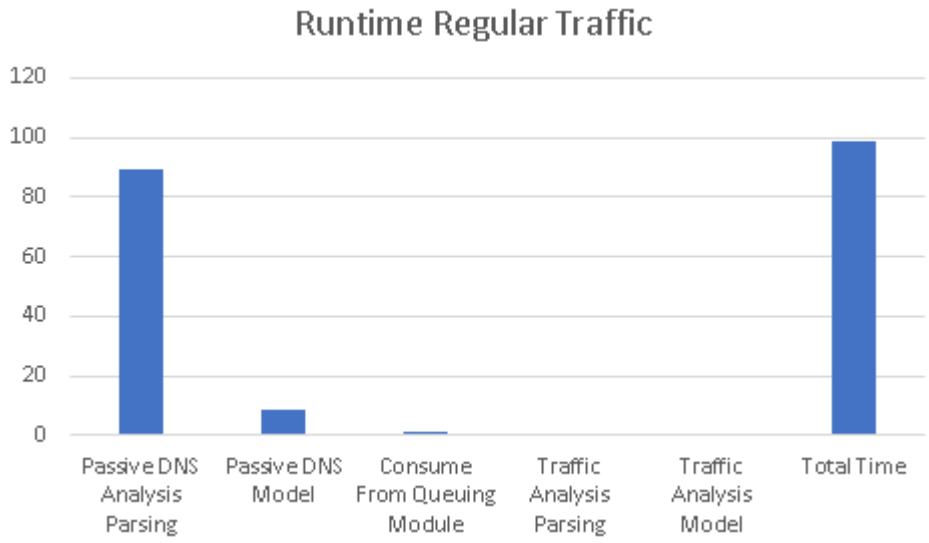


Figure 5.5: Runtime Regular Traffic Performance

Trying to reach the main problem of the delay on the Passive DNS performance we concluded that the amount of possible requests realized in the CIRCL database have a quota limit which don't allows us to execute an extensive check in DNS packets captured with a 0.5 seconds delay between each access, this may not be so efficient for a IDS system which required a near real time analysis, causing the delay presented before.

So to present a system which can function in a real world scenario we can isolate the general traffic analysis module an get a result of 1.3 seconds as can be seen on image 5.6, which bring an efficient form to a real world scenario, almost reaching a real time performance, only having a 1 second delay to consume the data from the scalable architecture

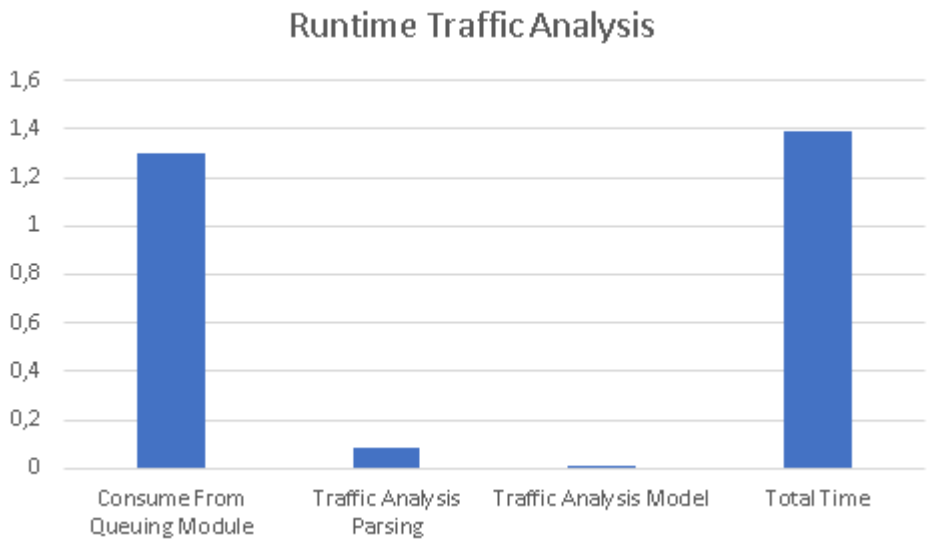


Figure 5.6: Runtime Traffic only Performance

5.7 Real Case Scenario

In this section will be presented an example of a full process execution of the attack been executed, as well as, the process handle each part of it.

To starts we prepared a regular host and a attack host on the SDN network. As we can see in the image 5.7, the regular host has the IP 20.1.1.2 and the image 5.8 shows the attacker host, with IP 20.1.1.4. Checking in the Opendaylight on image 5.9, it is possible to see by the Ether (Hardware Address) on the images that the hosts are up and the network connected to the SDN controller.

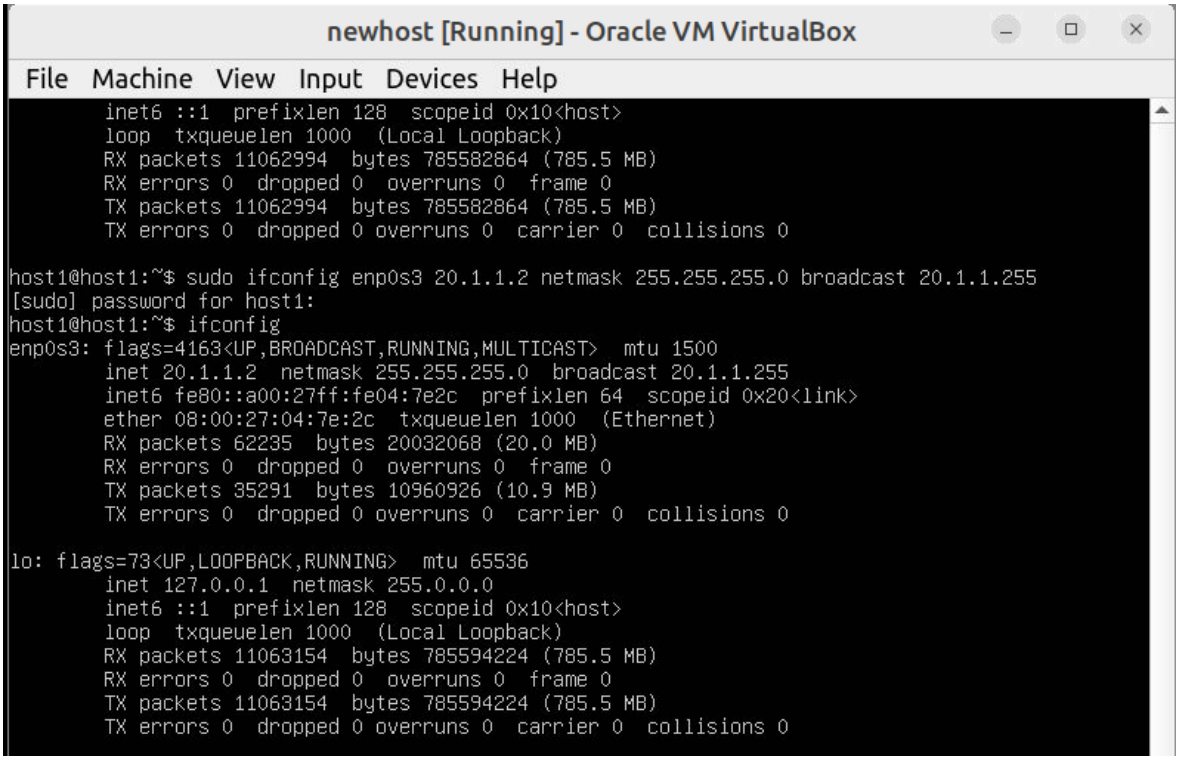


Figure 5.7: Real Scenario Regular Host

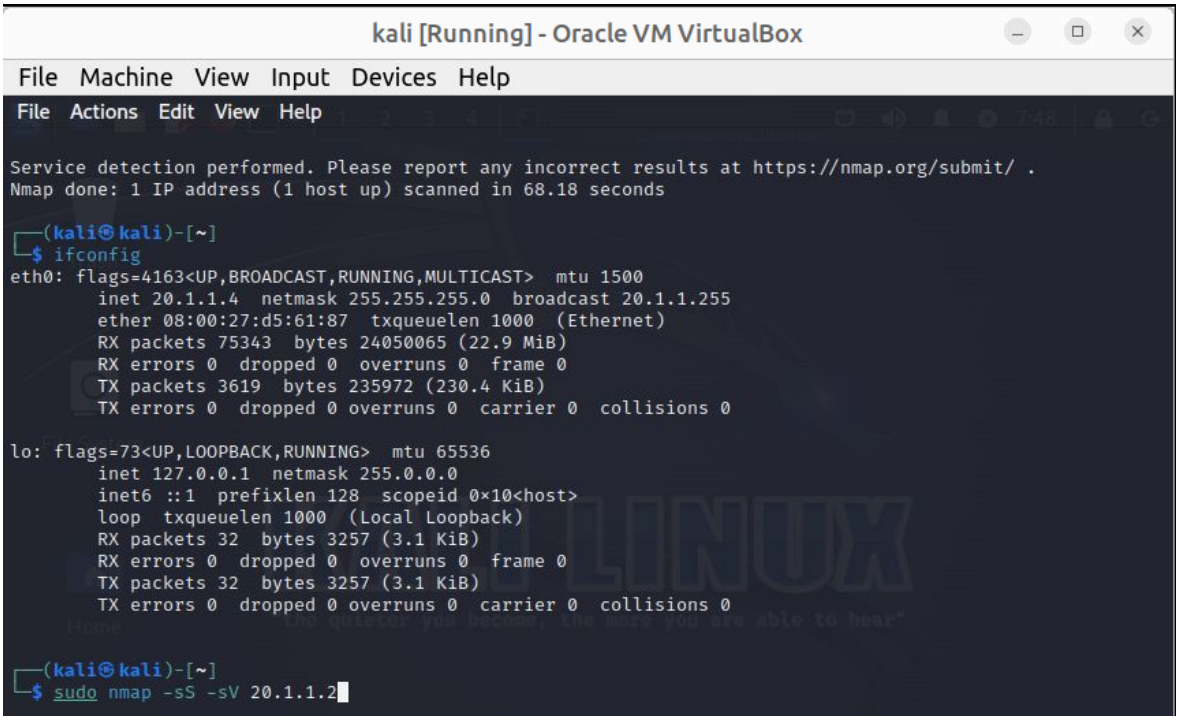


Figure 5.8: Real Scenario Kali Host

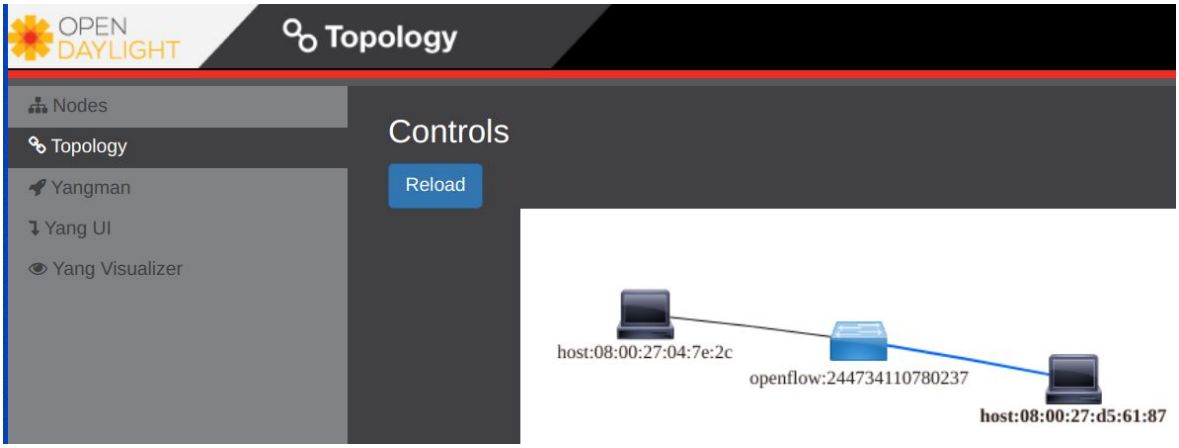


Figure 5.9: Real Scenario Topology

With the scenario set and ready to go we run the capture module code (appendix E.3) as it was explained on the diagram 4.6 of section 4.4. When the capture start, a simple message is displayed on the terminal indicating the capture configuration and that it has started (image 5.10). It's also possible to turn of the module by pressing CrtlC which will bring a little report about the amount of packets that were captured and transmitted to the queuing module.

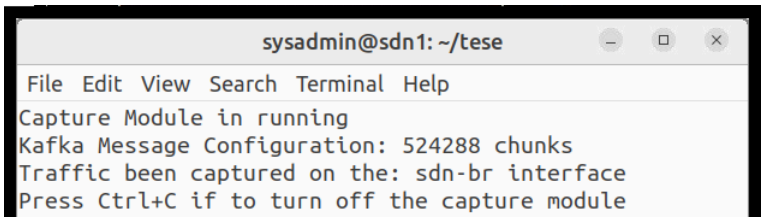


Figure 5.10: Real Scenario - Capture Module

As showed by the last line of the terminal in the image 5.8 the type of attack we executed in this example is a port mapping technique, using the NMAP tool provided in the Kali Linux OS.

In the Analysis Module we can see the packets been consumed and been processed. In the image 5.11, it is possible to see how the code received the raw data sent by the queuing module, and through the use of the PyPacker library as explained in the previous chapter, all the essential data coming from these packets is properly categorized and stored in a

CSV file. File that will be used by the AI model to predict if any of these are malicious or not. The output of this processing and extraction procedure can be visualized in the image 5.12.

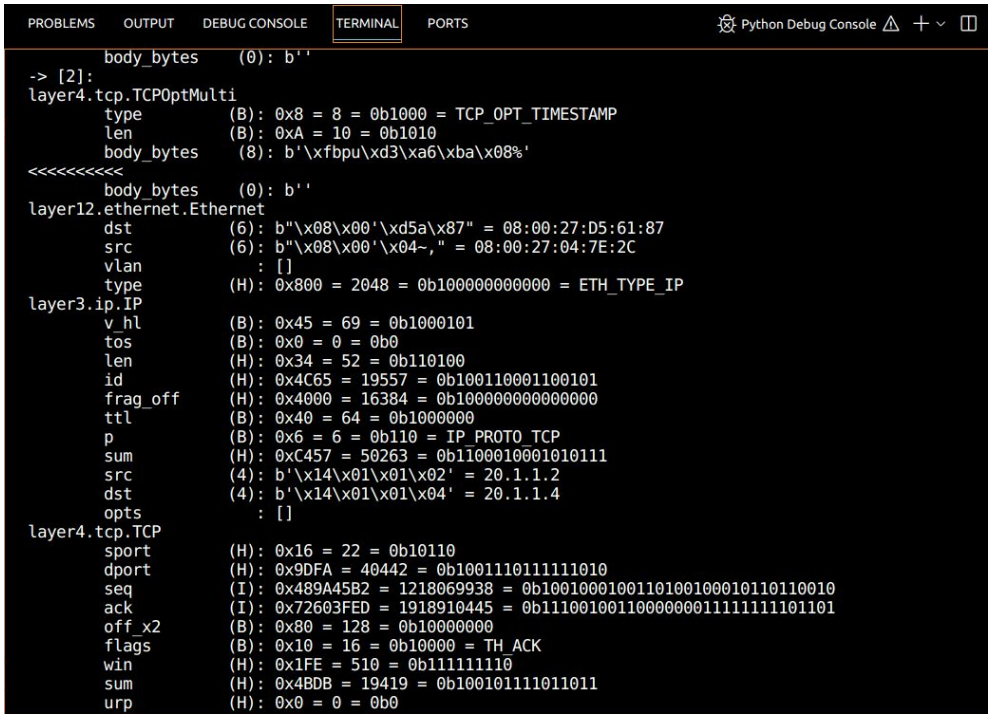
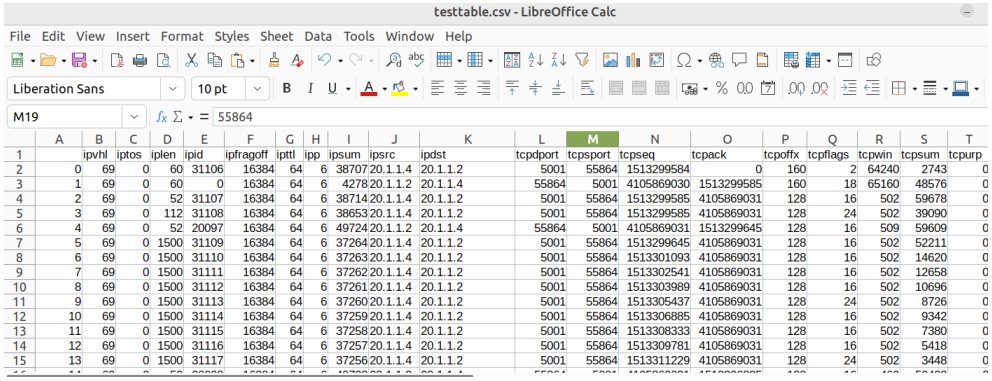


Figure 5.11: Real Scenario - Analysis Module - Raw Packets



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1		ipvhl	iptos	iplen	ipid	ipfragoff	ipttl	ipp	ipsum	ipsrc	ipdst	tcpdport	tcpsport	tcpseq	tcpack	tcpoffx	tcpflags	tcpwin	tcpsum	tcpurp
2	0	69	0	60	31106	16384	64	6	38707	20.1.1.4	20.1.1.2	5001	55864	1513299584	0	160	2	64240	2743	0
3	1	69	0	60	0	16384	64	6	4278	20.1.1.2	20.1.1.4	55864	5001	4105869030	1513299585	160	18	65160	48576	0
4	2	69	0	52	31107	16384	64	6	38714	20.1.1.4	20.1.1.2	5001	55864	1513299585	4105869031	128	16	502	59678	0
5	3	69	0	112	31108	16384	64	6	38653	20.1.1.4	20.1.1.2	5001	55864	1513299585	4105869031	128	24	502	39090	0
6	4	69	0	52	20097	16384	64	6	49724	20.1.1.2	20.1.1.4	55864	5001	4105869031	1513299645	128	16	509	59609	0
7	5	69	0	1500	31109	16384	64	6	37264	20.1.1.4	20.1.1.2	5001	55864	1513299645	4105869031	128	16	502	52211	0
8	6	69	0	1500	31110	16384	64	6	37263	20.1.1.4	20.1.1.2	5001	55864	1513301093	4105869031	128	16	502	14620	0
9	7	69	0	1500	31111	16384	64	6	37262	20.1.1.4	20.1.1.2	5001	55864	1513302541	4105869031	128	16	502	12658	0
10	8	69	0	1500	31112	16384	64	6	37261	20.1.1.4	20.1.1.2	5001	55864	1513303989	4105869031	128	16	502	10696	0
11	9	69	0	1500	31113	16384	64	6	37260	20.1.1.4	20.1.1.2	5001	55864	1513305437	4105869031	128	24	502	8726	0
12	10	69	0	1500	31114	16384	64	6	37259	20.1.1.4	20.1.1.2	5001	55864	1513306885	4105869031	128	16	502	9342	0
13	11	69	0	1500	31115	16384	64	6	37258	20.1.1.4	20.1.1.2	5001	55864	1513308333	4105869031	128	16	502	7380	0
14	12	69	0	1500	31116	16384	64	6	37257	20.1.1.4	20.1.1.2	5001	55864	1513309781	4105869031	128	16	502	5418	0
15	13	69	0	1500	31117	16384	64	6	37256	20.1.1.4	20.1.1.2	5001	55864	1513311229	4105869031	128	24	502	3448	0

Figure 5.12: Real Scenario - Analysis Module - Processed CSV

Now in the processing of Passive DNS packets, the final CSV is different, because of the field required to be analyzed. First, the previous packets that contain DNS protocols

are split from the others, and then a new process is made to extract the information. As it can be seen on image 5.13, it starts with decoding and identifying the DNS query responsible for that packet, and next it starts a query on the CIRCL database bringing the information available for that specific DNS, where all of the results of the queries are stored for the next step.

With the CIRCL information it's necessary to execute the process of calculating some of the characteristics presented before on section 4.6.2, turning the information to the database into a full characterization of each domain found on the traffic. The result is stored in another CSV file as presented on image 5.14, that will be used in the AI model, classifying as malicious or not.

```
sysadmin@sdn1:~/dns$ python3 passive.py
picaxe.com
['picaxe.com', ' NS', ' ns1.gradwellcloud.com', ' 1483609592', ' 1483609592']
['picaxe.com', ' NS', ' ns2.gradwellcloud.com', ' 1483609592', ' 1483609592']
['picaxe.com', ' NS', ' ns2.pickaweb.co.uk', ' 1540702105', ' 1540702105']
['picaxe.com', ' NS', ' ns1.pickaweb.co.uk', ' 1540702105', ' 1540702105']
['picaxe.com', ' A', ' 5.152.176.134', ' 1483609592', ' 1540702105']
picaxe.com
['picaxe.com', ' NS', ' ns1.gradwellcloud.com', ' 1483609592', ' 1483609592']
['picaxe.com', ' NS', ' ns2.gradwellcloud.com', ' 1483609592', ' 1483609592']
['picaxe.com', ' NS', ' ns2.pickaweb.co.uk', ' 1540702105', ' 1540702105']
['picaxe.com', ' NS', ' ns1.pickaweb.co.uk', ' 1540702105', ' 1540702105']
['picaxe.com', ' A', ' 5.152.176.134', ' 1483609592', ' 1540702105']
ajax.googleapis.com
```

Figure 5.13: Real Scenario - Analysis Module - Passive DNS Process

Final_Info.csv - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

Liberation Sans 10 pt

A1 Domain

	A	B	C	D	E	F	G	H	I	J
1	Domain	len_domain	max_len_labels	character_entropy	number_numeric	ratio_numerical_c	max_len_cont	max_len_cont	max_len_cont_	ratio_vowels
2	vk.cc	4	2	1.5	0	0.0	0	2	2	0.0
3	prgrams.com	10	7	2.9219280948873623	0	0.0	0	7	1	0.2
4	network.adsmarket.com	19	9	3.5110854081804272	0	0.0	0	9	1	0.3157894736842105
5	thumbs.redtube.com	16	7	3.375	0	0.0	0	7	1	0.3125
6	cityofwestmiamipublicservices.com	32	29	3.816428031846024	0	0.0	0	29	1	0.375
7	404.ragecash.com	14	8	3.3787834934861 76	3	0.2142857142857	3	8	1	0.2857142857142857
8	voyageclub.ru	12	10	3.4182958340544896	0	0.0	0	10	1	0.4166666666666667
9	fundacionelfortin.org	20	17	3.5464394446710147	0	0.0	0	17	1	0.4
10	yqmjnbmxzfz.com	14	11	3.0930692077718893	0	0.0	0	11	1	0.07142857142857142
11	soulanimtech.com	15	12	3.5068905956085183	0	0.0	0	12	1	0.4
12	wifelovers.com	13	10	3.3927474104487847	0	0.0	0	10	1	0.38461538461538464

Figure 5.14: Real Scenario - Analysis Module - Passive DNS Characterization

Now with all the information properly processed, the Analyzer execute the prediction based on each field of the information, and if any packet contains a malicious behavior he will notify the user by presenting a message containing the host responsible for that traffic (image 5.15).

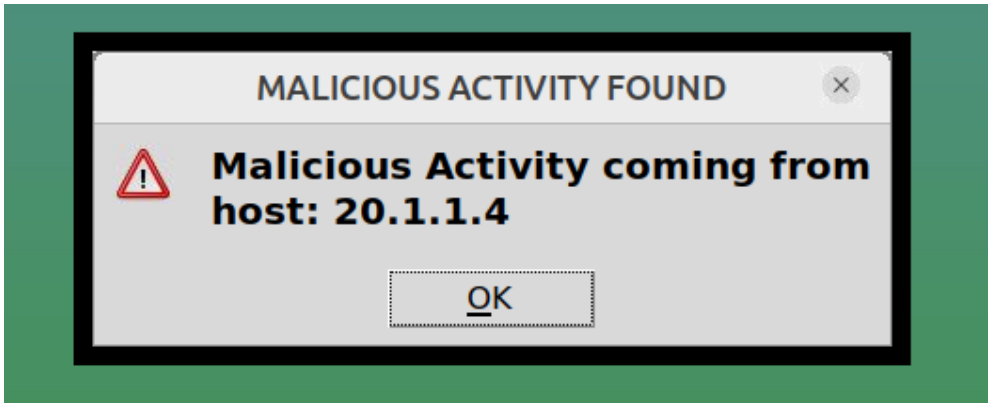


Figure 5.15: Real Scenario - Analysis Module - Warning Result

Chapter 6

Conclusions and future work

This work was developed jointly with the "CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)" which provides the structure to develop the described architecture of this work. The solution found is based on the studies identified in the systematic review, unifying knowledge to bring the resolution of this work.

Using the researches made previously by scientific community, the deployment of the SDN network was a success where it's possible to add more virtual machines in the network to simulate and work with any scenario wanted. Also based on Oliveira and Pedrosa[16] solution to a scalable topology, was possible to link our capture module and SDN topology to a more complex architecture were all data could be stored, replicated and consumed whenever needed.

Taking into consideration our experiments made in the last section, it's possible to see that the capture solution performed really well, keeping a very low rate of packet loss by varying between 0,002% and 0,005%, where in cases of intense network traffic it reaches 0,05%. Grouping and sending almost all the packets that passes through the sniffer.

Now about the scalable topology, it worked successfully where the Kafka nodes stored correctly the traffic captured in the SDN topology, reaching a 0.25 second in the transmission of the data, and the consume rate reaching a 1.3 seconds. These results brings to us the successful usage of a scalable system for a real scenario which new

nodes are always welcome due the growing number of possible hosts in the network.

Talking on the solution proposed for the Intrusion Detection System, the main module for analysing the general traffic of the hosts get an accuracy of 96% on finding the malicious packets, during a period of time of 0.94 seconds (Traffic Parsing + Traffic Analysis). When including the time of consumption reaching 1.3 seconds in the total process duration. The Passive DNS solution shows promising results where the analysis module on identifying malicious DNS get a positive result of 94%, showing reliability but for it's usage in a Intrusion Detection System would be necessary an improvement in the maximum quota available for the requests in the CIRCL database for fastest responses.

6.1 Future Work

As a first step on researching the possibilities of providing security to a SDN architecture, this work showed promising results, with a good performance in capturing and consuming traffic and also providing a scalable system for this solution.

This work is the first of a large variety of possibilities for the future. It only shows a small part of the power and the potential of the SDN and more extensive cases that a IDS can cover. With the promising results of this work, in the future will be needed more test and work to different scenarios.

To start this solution needs to be tested in a real SDN implementation scenario, with modern physical switches that allow the Openflow communication for the SDN deployment, which will be needed to do a comparison between the throughput provided by this work in a virtual scenario and the real scenario, proving the real performance of a real SDN deployment.

In the AI model side, the number of possible attacks and malicious situations in a real scenario are huge, which bring us the need of works that can amplify the amount of attacks detected by the model. Collecting more data to the Datasets gathered for the AI training, since the attacks presented here are just the beginning of a possible attack, and don't include the more sophisticated approaches by the attacker.

The Passive DNS solution presented a positive result on detecting malicious DNS activities, but as previously stated the performance of the CIRCL access limited the project on the response time. Trying reach responsible for the CIRCL project and gather more information about the quota available for Passive DNS requests is necessary for a more robust and effective solution, and new researches could actual find more effective approaches to increase the Passive DNS solution.

Also the CIRCL database has a big opportunity for a future work in a Passive SSL solution, which their libraries include functions to execute Passive SSL approaches which can be a good feature for the future of the IDS system on consulting the truthfulness of the SSLs captured in the traffic.

Bibliography

[1] M. Mousa, A. M. Bahaa-Eldin and M. Sobh, "Software defined networking concepts and challenges," in *2016 11th International Conference on Computer Engineering Systems (ICCES)*, 2016, pp. 79–90. DOI: 10.1109/ICCES.2016.7821979.

[2] O. Starkova, K. Herasymenko, K. Nikolchev, O. Zelikovska, A. Bulgakova and N. Solovey, "Virtualization and programmability in modern networks in the context of sdn concept," in *2022 IEEE 4th International Conference on Advanced Trends in Information Theory (ATIT)*, 2022, pp. 204–207. DOI: 10.1109/ATIT58178.2022.10024178.

[3] U. Tupakula, K. K. Karmakar, V. Varadharajan and B. Collins, "Implementation of techniques for enhancing security of southbound infrastructure in sdn," in *2022 13th International Conference on Network of the Future (NoF)*, 2022, pp. 1–5. DOI: 10.1109/NoF55974.2022.9942644.

[4] M. Byun, Y. Lee and J.-Y. Choi, "Risk and avoidance strategy for blocking mechanism of sdn-based security service," in *2019 21st International Conference on Advanced Communication Technology (ICACT)*, 2019, pp. 187–190. DOI: 10.23919/ICACT.2019.8701887.

[5] N. Feamster, J. Rexford and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014, ISSN: 0146-4833. DOI: 10.1145/2602204.2602219. [Online]. Available: https://doi.org/10.1145/2602204.2602219.

[6] A. Malishevskiy, D. Gurkan, L. Dane, R. Narisetty, S. Narayan and S. Bailey, "Openflow-based network management with visualization of managed elements," in *2014 Third GENI Research and Educational Experiment Workshop*, 2014, pp. 73–74. DOI: 10.1109/GREE.2014.21.

[7] N. El Moussaid, A. Toumanari and M. El Azhari, "Security analysis as software-defined security for sdn environment," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, 2017, pp. 87–92. DOI: 10.1109/SDS.2017.7939146.

[8] 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, Oct. 2016, ISSN: 1383-469X. DOI: 10.1007/s11036-016-0676-x. [Online]. Available: https://doi.org/10.1007/s11036-016-0676-x.

[9] C. Birkinshaw, E. Rouka and V. G. Vassilakis, "Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks," *J. Netw. Comput. Appl.*, vol. 136, no. C, pp. 71–85, Jun. 2019, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2019.03.005. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.03.005.

[10] 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, pp. 37–44. DOI: 10.1109/NETWKS.2016.7751150.

[11] M. M. Isa and L. Mhamdi, "Native sdn intrusion detection using machine learning," in *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*, 2020, pp. 1–7. DOI: 10.1109/ComNet47917.2020.9306093.

[12] M. Tavallaee, E. Bagheri, W. Lu and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[13] R. M. A. Ujjan, Z. Pervez and K. Dahal, "Snort based collaborative intrusion detection system using blockchain in sdn," in *2019 13th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, 2019, pp. 1–8. DOI: 10.1109/SKIMA47702.2019.8982413.

[14] R. F. Pratama, N. A. Suwastika and M. A. Nugroho, "Design and implementation adaptive intrusion prevention system (ips) for attack prevention in software-defined network (sdn) architecture," in *2018 6th International Conference on Information and Communication Technology (ICoICT)*, 2018, pp. 299–304. DOI: 10.1109/ICoICT.2018.8528735.

[15] P. Ohri, S. G. Neogi and S. K. Muttoo, "Security analysis of open source sdn (odl and onos) controllers," in *2023 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, 2023, pp. 1–4. DOI: 10.1109/SCEECS57921.2023.10063108.

[16] R. C. d. Oliveira, *Near real-time network analysis for the identification of malicious activity — hdl.handle.net*, http://hdl.handle.net/10198/24947, [Accessed 25-May-2023], 2021.

[17] F. J. Badaró V. Neto, C. J. Miguel, A. C. d. S. de Jesus and P. N. Sampaio, "SDN Controllers - A Comparative approach to Market Trends," in *9th International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2021)*, ser. Proc. of the 9th International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2021), Rafael Tolosana Calasanz, General Chair and Gabriel Gonzalez-Castañé, TPC Co-Chair and Nazim Agoulmine, Steering Committee Chair, Zaragoza, Spain, Feb. 2021, pp. 48–51. DOI: 10.48545/advance2021-shortpapers-3. [Online]. Available: https://hal.science/hal-03133692.

[18] *Opendaylight — developer.cisco.com*, https://developer.cisco.com/site/opendaylight/, [Accessed 07-10-2023].

[19] M. P. Contributors, *Mininet Walkthrough - Mininet — mininet.org*, http://mininet.org/walkthrough/, [Accessed 07-10-2023].

[20] *Proxmox VE Documentation Index — pve.proxmox.com*, https://pve.proxmox.com/pve-docs/, [Accessed 07-10-2023].

[21] Y. A. Farrukh, I. Khan, S. Wali, D. Bierbrauer, J. A. Pavlik and N. D. Bastian, "Payload-byte: A tool for extracting and labeling packet capture files of modern network intrusion detection datasets," in *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, 2022, pp. 58–67. DOI: 10.1109/BDCAT56447.2022.00015.

[22] K. Peddireddy, "Streamlining enterprise data processing, reporting and realtime alerting using apache kafka," in *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*, 2023, pp. 1–4. DOI: 10.1109/ISDFS58141.2023.10131800.

[23] *Apache Kafka — kafka.apache.org*, https://kafka.apache.org/documentation/, [Accessed 07-10-2023].

[24] Y. Tian and X. Yu, "Trustworthiness study of hdfs data storage based on trustworthiness metrics and kms encryption," in *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, 2021, pp. 962–966. DOI: 10.1109/ICPECA51329.2021.9362537.

[25] *Hadoop x2013; Apache Hadoop 3.3.6 — hadoop.apache.org*, https://hadoop.apache.org/docs/stable/, [Accessed 07-10-2023].

[26] *3.12.0 Documentation — docs.python.org*, https://docs.python.org/3/, [Accessed 07-10-2023].

[27] T.-T.-H. Le, H. Kim, H. Kang and H. Kim, "Classification and explanation for intrusion detection system based on ensemble trees and shap method," *Sensors*, vol. 22, no. 3, 2022, ISSN: 1424-8220. DOI: 10.3390/s22031154. [Online]. Available: https://www.mdpi.com/1424-8220/22/3/1154.

[28] T. Zebin, S. Rezvy and Y. Luo, "An explainable ai-based intrusion detection system for dns over https (doh) attacks," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2339–2349, 2022. DOI: 10.1109/TIFS.2022.3183390.

[29] R. Kumar, P. Kumar, R. Tripathi, G. P. Gupta, S. Garg and M. M. Hassan, "A distributed intrusion detection system to detect ddos attacks in blockchain-enabled iot network," *Journal of Parallel and Distributed Computing*, vol. 164, pp. 55–68, 2022, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2022.01.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731522000351.

[30] S. Afroz, S. Ariful Islam, S. Nawer Rafa and M. Islam, "A two layer machine learning system for intrusion detection based on random forest and support vector machine," in *2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, 2020, pp. 300–303. DOI: 10.1109/WIECON-ECE52138.2020.9397945.

[31] G. Xuanzhen, P. Zulie and C. Yuanchao, "Application of passive dns in cyber security," in *2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, 2020, pp. 257–259. DOI: 10.1109/ICPICS50287.2020.9202344.

[32] C. Han and Y. Zhang, "Clean : An approach for detecting benign domain names based on passive dns traffic," in *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*, 2017, pp. 343–346. DOI: 10.1109/ICCSNT.2017.8343715.

[33] *CIRCL &xBB; CIRCL – Computer Incident Response Center Luxembourg – CSIRT – CERT — circl.lu*, https://www.circl.lu, [Accessed 27-10-2023].

[34] *GitHub - mike01/pypacker: :package: The fastest and simplest packet manipulation lib for Python — github.com*, https://github.com/mike01/pypacker, [Accessed 27-10-2023].

[35] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] *Common Configuration Issues &x2014; Open vSwitch 3.2.90 documentation — docs.openvswitch.org*, https://docs.openvswitch.org/en/latest/faq/issues/, [Accessed 28-10-2023].

[37] *Kali Docs | Kali Linux Documentation — kali.org*, https://www.kali.org/docs/, [Accessed 28-10-2023].

[38] I. Khalil, T. Yu and B. Guan, "Discovering malicious domains through passive dns data graph analysis," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16, Xi'an, China: Association for Computing Machinery, 2016, pp. 663–674, ISBN: 9781450342339. DOI: 10.1145/2897845.2897877. [Online]. Available: https://doi.org/10.1145/2897845.2897877.

[39] Z. Liu, Y. Zeng, P. Zhang, J. Xue, J. Zhang and J. Liu, "An imbalanced malicious domains detection method based on passive dns traffic analysis," *Security and Communication Networks*, vol. 2018, p. 6 510 381, Jun. 2018, ISSN: 1939-0114. DOI: 10.1155/2018/6510381. [Online]. Available: https://doi.org/10.1155/2018/6510381.

Appendix A

Original dissertation proposal

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Thesis Proposal

Master in Informatics

2021/2022

Title: SDN IDS

Supervisor (IPB): Tiago Pedrosa (riftman@ipb.pt) | Nuno Rodrigues (nuno@ipb.pt)

Supervisor (UTFPR): Augusto Foronda (foronda@utfpr.edu.br)

Student: Vinicius Lopes (a49816@alunos.ipb.pt)

Context:

Nowadays with the increase of the companies' information and communication infrastructure, having a high number of servers, devices and networks, companies seek to migrate their communication infrastructure technology to a more flexible one. These last years the trend is in having a virtualized networks,software-defined networks, this way, with the separation of the network from the hardware the management of it, is much more practical and new efforts are possible, as it is all done through software at a global level. Also, this is the reality of the cloud services that multiple companies provide and many acquire. Like the traditional networks, this recent technology also needs to be assessed to find malicious activity on the network. This need to be performed differently as there is a lack of hardware devices to connect to and capture the network traffic for further analysis.

Goal

This project will be directly integrated into an ongoing investigation project designated as “CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)”. The research project consists of a distributed intrusion detection system (IDS) with the analysis of the network data in near real-time. The connection of this dissertation with the research project would be in the implementation of a controller able to capture the network traffic from a set of selected virtual devices and to perform certain tasks accordingly to the output of the intrusion detection system like real-time blockage of certain traffic. The second objective would consist of the implementation of new analysis applications for the IDS to improve the detection of malicious activity.

Prerequisites

Tasks

This work comprises the execution of the following stages:

1. Literature review and study of related works for familiarization and identification of requirements, concepts and technologies that will be used for the development of the work;
2. Systematic review of a set of solutions to determine which solution will fit the needs;
3. Definition of the system architecture and components;
4. Develop and implementation of the sniffer solution;
5. Deploy the SDN solution;
6. Creation of a scenario with several virtual machines on the SDN simulating normal and abnormal communication between server and client machines;
7. Propose new analysis applications for the IDS to improve the detection of malicious activity;
8. Experiments, analysis of results and possible improvements;
9. Documentation and writing of the dissertation.

Infrastructure and resources required

Laptop and virtual machines

Appendix B

Opendaylight Configuration

B.1 Opendaylight Installation Script

```
#!/bin/bash

# Variables
ODL_VERSION="0.8.4"
ODL_URL="https://downloads.opendaylight.org/odl/opendaylight-
   $ODL_VERSION.zip"
INSTALL_DIR="/opt/opendaylight"

# Prerequisites
sudo apt-get update
sudo apt-get install -y openjdk-8-jre unzip

# Download and extract OpenDaylight
mkdir -p $INSTALL_DIR
cd $INSTALL_DIR
wget $ODL_URL
unzip opendaylight-$ODL_VERSION.zip
```

```
rm opendaylight-$ODL_VERSION.zip

# Create a symbolic link to the current ODL version (optional)
ln -s opendaylight-$ODL_VERSION current

# Start OpenDaylight
cd $INSTALL_DIR/current/bin
./karaf

# OpenDaylight should now be running. You can access the Karaf
   console via SSH (if enabled) or a web browser (http://your-
   server-ip:8181). Default credentials are username: "admin" and
    password: "admin".
```

B.2 Opendaylight Configuration Script

```

odl-cluster-data {
  akka {
    remote {
      artery {
        enabled = off
        canonical.hostname = "127.0.0.1"
        canonical.port = 2550
      }
      netty.tcp {
        hostname = "127.0.0.1"
        port = 2550
      }
```

```
14        # when under load we might trip a false positive on the
              failure detector
15        # transport-failure-detector {
16          # heartbeat-interval = 4 s
17          # acceptable-heartbeat-pause = 16s
18        # }
19      }
20
21      cluster {
22        # Remove ".tcp" when using artery.
23        seed-nodes = ["akka.tcp://opendaylight-cluster-data@127
              .0.0.1:2550"]
24
25        roles = [
26          "member-1"
27        ]
28
29      }
30
31      persistence {
32        # By default the snapshots/journal directories live in
              KARAF_HOME. You can choose to put it somewhere else by
33        # modifying the following two properties. The directory
              location specified may be a relative or absolute path.
34        # The relative path is always relative to KARAF_HOME.
35
36        # snapshot-store.local.dir = "target/snapshots"
37        # journal.leveldb.dir = "target/journal"
38
39        journal {
```

```
40          leveldb {
41            # Set native = off to use a Java-only implementation of
                  leveldb.
42            # Note that the Java-only version is not currently
                 considered by Akka to be production quality.
43
44            # native = off
45          }
46        }
47      }
48    }
49  }
```

Appendix C

OpenVSwitch Configuration

C.1 OpenVSwitch Installation Script

```
#!/bin/bash

# Update the package repository
sudo apt-get update

# Install Open vSwitch and its utilities
sudo apt-get install -y openvswitch-switch openvswitch-common

# Start the Open vSwitch service
sudo systemctl start openvswitch-switch

# Enable Open vSwitch to start on boot
sudo systemctl enable openvswitch-switch

# Verify the installation
ovs-vsctl --version

```

```
# Optional: Create a bridge interface (replace "br0" and "eth0" with your desired names)
# sudo ovs-vsctl add-br br0
# sudo ovs-vsctl add-port br0 eth0

# Optional: Configure bridge for management (replace "10.0.0.10/24" and "br0" with your desired IP and bridge)
# sudo ifconfig br0 10.0.0.10/24 up

# Optional: Ensure the bridge survives reboots
# sudo ovs-vsctl set bridge br0 other_config="stp-enable=true"
# sudo ovs-vsctl set bridge br0 other_config="forward-bpdu=true"

# Optional: Add a default route (replace "10.0.0.1" with your gateway)
# sudo ip route add default via 10.0.0.1 dev br0

# Optional: Enable IP forwarding (if needed)
# echo 'net.ipv4.ip_forward=1' | sudo tee -a /etc/sysctl.conf
# sudo sysctl -p

# Optional: Add OVS to your firewall configuration (if you're using iptables)
# iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT

# Confirm OVS bridge and network configuration
# sudo ovs-vsctl show

echo "Open vSwitch has been installed and configured."

```

```
# You may need to restart your networking service to apply the changes
# sudo service networking restart
```

C.2 OpenVSwitch Bridge Configuration

```
4e0985ca-d77c-4f2f-b610-cd881464081d
    Bridge sdn-br
        Controller "tcp:192.168.199.231:6633"
            is_connected: true
        Port sdn-br
            Interface sdn-br
                type: internal
        Port host4
            Interface host4
        Port host1
            Interface host1
        Port host2
            Interface host2
        Port host3
            Interface host3
    ovs_version: "2.17.5"
```

Appendix D

Interfaces Configuration

D.1 Interfaces example with 4 hosts

```
ens18: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.2.60  netmask 255.255.255.0  broadcast
           10.1.2.255
        inet6 2001:690:22c0:8a03:6cb6:bc37:a14f:8c70  prefixlen
           64  scopeid 0x0<global>
        inet6 fe80::e5f:9075:5eb:7e5b  prefixlen 64  scopeid 0x20
           <link>
        inet6 2001:690:22c0:8a03:e42c:e388:c874:224f  prefixlen
           64  scopeid 0x0<global>
        inet6 2001:690:22c0:8a03:2d36:1c15:b41c:f033  prefixlen
           64  scopeid 0x0<global>
        inet6 2001:690:22c0:8a03:e775:7443:ce3a:e439  prefixlen
           64  scopeid 0x0<global>
        inet6 2001:690:22c0:8a03:911d:3831:269e:74c6  prefixlen
           64  scopeid 0x0<global>
        inet6 2001:690:22c0:8a03:3bcf:1db:d130:e802  prefixlen 64
             scopeid 0x0<global>
```

```
10          inet6 2001:690:22c0:8a03:e91b:a986:8b9:875b  prefixlen 64
                scopeid 0x0<global>
11          inet6 2001:690:22c0:8a03:db19:a7c6:3d47:f4b0  prefixlen
               64  scopeid 0x0<global>
12          ether 62:ca:40:33:78:0d  txqueuelen 1000  (Ethernet)
13          RX packets 16805567  bytes 4796017825 (4.7 GB)
14          RX errors 0  dropped 30657  overruns 0  frame 0
15          TX packets 11724849  bytes 7957569358 (7.9 GB)
16          TX errors 0  dropped 0 overruns 0  carrier 0  collisions
               0
17
18  ens19: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
19          inet 192.168.199.231  netmask 255.255.0.0  broadcast
               192.168.255.255
20          inet6 fe80::288d:7eee:9395:9d10  prefixlen 64  scopeid 0
               x20<link>
21          ether 0e:7a:dc:3c:3e:12  txqueuelen 1000  (Ethernet)
22          RX packets 115595964  bytes 16282521332 (16.2 GB)
23          RX errors 0  dropped 1065876  overruns 0  frame 0
24          TX packets 535339  bytes 57421511 (57.4 MB)
25          TX errors 0  dropped 0 overruns 0  carrier 0  collisions
               0
26
27  host1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
28          inet6 fe80::d09f:77ff:fefe:5591  prefixlen 64  scopeid 0
               x20<link>
29          ether d2:9f:77:fe:55:91  txqueuelen 1000  (Ethernet)
30          RX packets 0  bytes 0 (0.0 B)
31          RX errors 0  dropped 0  overruns 0  frame 0
32          TX packets 0  bytes 0 (0.0 B)
```

```
33          TX errors 0  dropped 201425 overruns 0  carrier 0
               collisions 0
34
35  host2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
36          inet6 fe80::7ce1:eff:fecb:eafc  prefixlen 64  scopeid 0
               x20<link>
37          ether 7e:e1:0e:cb:ea:fc  txqueuelen 1000  (Ethernet)
38          RX packets 0  bytes 0 (0.0 B)
39          RX errors 0  dropped 0  overruns 0  frame 0
40          TX packets 0  bytes 0 (0.0 B)
41          TX errors 0  dropped 164856 overruns 0  carrier 0
               collisions 0
42
43  host3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
44          inet6 fe80::dcf9:35ff:fe61:b2c7  prefixlen 64  scopeid 0
               x20<link>
45          ether de:f9:35:61:b2:c7  txqueuelen 1000  (Ethernet)
46          RX packets 0  bytes 0 (0.0 B)
47          RX errors 0  dropped 0  overruns 0  frame 0
48          TX packets 0  bytes 0 (0.0 B)
49          TX errors 0  dropped 111906 overruns 0  carrier 0
               collisions 0
50
51  host4: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
52          inet6 fe80::ecf2:8bff:fea1:1020  prefixlen 64  scopeid 0
               x20<link>
53          ether ee:f2:8b:a1:10:20  txqueuelen 1000  (Ethernet)
54          RX packets 0  bytes 0 (0.0 B)
55          RX errors 0  dropped 0  overruns 0  frame 0
56          TX packets 0  bytes 0 (0.0 B)
```

```
57          TX errors 0  dropped 74998 overruns 0  carrier 0
               collisions 0
58
59  lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
60          inet 127.0.0.1  netmask 255.0.0.0
61          inet6 ::1  prefixlen 128  scopeid 0x10<host>
62          loop  txqueuelen 1000  (Local Loopback)
63          RX packets 30038751  bytes 8228591010 (8.2 GB)
64          RX errors 0  dropped 0  overruns 0  frame 0
65          TX packets 30038751  bytes 8228591010 (8.2 GB)
66          TX errors 0  dropped 0 overruns 0  carrier 0  collisions
               0
67
68  sdn-br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
69          inet 20.1.1.1  netmask 255.255.255.0  broadcast
               20.1.1.255
70          inet6 fe80::dc95:99ff:febc:cb4d  prefixlen 64  scopeid 0
               x20<link>
71          ether de:95:99:bc:cb:4d  txqueuelen 1000  (Ethernet)
72          RX packets 507783  bytes 72858918 (72.8 MB)
73          RX errors 0  dropped 404080  overruns 0  frame 0
74          TX packets 131142  bytes 27663795 (27.6 MB)
75          TX errors 0  dropped 0 overruns 0  carrier 0  collisions
               0
```

Appendix E

Code Developed

E.1 Kafka Administration Code

```
from confluent_kafka.admin import AdminClient, NewTopic

BOOTSTRAP_SERVERS = 'kafka1.alunos.local:9093,kafka2.alunos.local
   :9093,kafka3.alunos.local:9093,kafka4.alunos.local:9093'
CERTIFICATE_LOCATION = '/ssl/client.crt'
CERTIFICATE_KEY_LOCATION = '/ssl/client.key'
CA_LOCATION = '/ssl/alunos.crt'

admin = AdminClient({
    'bootstrap.servers': BOOTSTRAP_SERVERS,
    'security.protocol': 'SSL',
    'ssl.certificate.location': CERTIFICATE_LOCATION,
    'ssl.key.location': CERTIFICATE_KEY_LOCATION,
    'ssl.ca.location': CA_LOCATION
})


```

```
def create_topics(topics, num_partitions=1, replication_factor=1):
    if type(topics) is str:
        topics = [topics]

    new_topics = [NewTopic(topic, num_partitions=num_partitions,
       replication_factor=replication_factor) for topic in
                  topics]

    fs = admin.create_topics(new_topics)

    for topic, f in fs.items():
        try:
            f.result()
            print(f'Topic {topic} created')
        except Exception as e:
            print(f'Failed to create topic: {topic}: {e}')


def delete_topics(topics, operation_timeout=30):
    if type(topics) is list:
        fs = admin.delete_topics(topics, operation_timeout=
           operation_timeout)
    elif type(topics) is str:
        fs = admin.delete_topics([topics], operation_timeout=
           operation_timeout)
    else:
        return

    for topic, f in fs.items():
```

```
43          try:
44              f.result()
45              print(f'Topic {topic} deleted')
46          except Exception as e:
47              print(f'Failed to delete topic {topic}: {e}')
48
49
50  def get_topics():
51      """
52      :return: dictionary containing all the topics
53      """
54      metadata = admin.list_topics()
55
56      return metadata.topics
57
58
59  def topic_information(topic):
60      name = topic.topic
61      partitions = topic.partitions
62
63      print(f'name: {name}, partitions: {list(partitions.keys())}')
64
65
66  def list_topics():
67      topics = get_topics()
68
69      for b in topics:
70          topic_information(topics[b])
71
72
```

```
73  def get_brokers():
74      metadata = admin.list_topics()
75
76      return metadata.brokers
77
78
79  def broker_information(broker):
80      print(f'[{broker.id}] {broker.host}:{broker.port}')
81
82
83  def list_brokers():
84      brokers = get_brokers()
85
86      for b in brokers:
87          broker_information(brokers[b])
88
89  print("\n")
90  topic = get_topics()
91  print("List of Topics:")
92  print(topic)
93  print("\n")
94  topic = list_topics()
95  print("\n")
96  #delete_topics('packetcapture')
97  create_topics('packetcapture',4)
98  #create_topics('__consumer_offsets',50)
99  print("\n")
100 topic = get_topics()
101 print("List of Topics:")
102 print(topic)
```

```
print("\n")
topic = list_topics()
print("\n")
```

E.2 Kafka Configuration for Capture Code

```

import argparse

BOOTSTRAP_SERVERS = 'kafka1.alunos.local:9093,kafka2.alunos.local
    :9093,kafka3.alunos.local:9093,kafka4.alunos.local:9093'
CERTIFICATE_LOCATION = '/ssl/client.crt'
CERTIFICATE_KEY_LOCATION = '/ssl/client.key'
CA_LOCATION = '/ssl/alunos.crt'
PCAP_FILE_NAME = 'dump.pcap'
```

E.3 Capture Module Code

```

from confluent_kafka import Producer
from custom_configs import *
import subprocess
import threading
import datetime
import signal
import json
import time
import re
import os
```

```
12
13 CUSTOM_HEADER_SIZE = 24
14 CUSTOM_PACKET_HEADER_SIZE = 16
15
16 custom_producer = Producer({
17     'bootstrap.servers': CUSTOM_BOOTSTRAP_SERVERS,
18     'security.protocol': 'CUSTOM_SSL',
19     'ssl.certificate.location': CUSTOM_CERTIFICATE_LOCATION,
20     'ssl.key.location': CUSTOM_CERTIFICATE_KEY_LOCATION,
21     'ssl.ca.location': CUSTOM_CA_LOCATION
22 })
23
24 CUSTOM_FINISHED = False
25
26 custom_log_dir = 'custom_logs'
27 os.makedirs(custom_log_dir, exist_ok=True)
28 custom_date = datetime.datetime.now()
29 custom_log_filename = f'custom_logs/custom_producer_{custom_date.
     strftime("%Y-%m-%d_%H_%M_%S")}.json'
30
31 custom_capture_started = None
32 custom_capture_ended = None
33 custom_packets_captured = None
34 custom_packets_dropped = None
35 custom_packets_received_by_filter = None
36 custom_kafka_ended = None
37 custom_pcap_sh1 = None
38
39 def custom_signal_handler(sig, frame):
40     if custom_tcpdump_process is not None:
```

```
41          custom_tcpdump_process.terminate()
42
43  def custom_start_writer():
44      global custom_tcpdump_process, CUSTOM_FINISHED,
            custom_capture_ended, custom_capture_started,
            custom_packets_captured, \
45          custom_packets_dropped, custom_packets_received_by_filter
46
47      custom_capture_started = time.time()
48
49      custom_tcpdump_command = ['custom_tcpdump', CUSTOM_FILTER, '-
            i', CUSTOM_NIC, '-s', str(CUSTOM_SNAPLEN), '-w',
            CUSTOM_PCAP_FILE_NAME]
50      custom_tcpdump_process = subprocess.Popen(
            custom_tcpdump_command, stdout=subprocess.PIPE, stderr=
            subprocess.STDOUT, universal_newlines=True)
51      custom_tcpdump_process.wait()
52
53      custom_tcpdump_output = custom_tcpdump_process.stdout.read()
54      CUSTOM_FINISHED = True
55      custom_capture_ended = time.time()
56
57      # Parse statistics from custom_tcpdump output
58      custom_statistics = re.findall(r"\n[0-9]+",
            custom_tcpdump_output)
59      custom_packets_captured = int(custom_statistics[0])
60      custom_packets_received_by_filter = int(custom_statistics[1])
61      custom_packets_dropped = int(custom_statistics[2])
62      print("custom_tcpdump has finished")
63      print("Publishing the rest of the data...")
```

```
64
65 def custom_start_reader():
66     global custom_kafka_ended
67
68     custom_pcap_skipped = False
69     custom_pcap_file = open(CUSTOM_PCAP_FILE_NAME, 'rb')
70     custom_left_overs = b''
71
72     while True:
73         custom_chunk = custom_left_overs + custom_pcap_file.read(
               CUSTOM_MESSAGE_SIZE - len(custom_left_overs))
74         custom_left_overs = b''
75
76         if custom_chunk == b'' and CUSTOM_FINISHED:
77             break
78
79         if custom_chunk == b'':
80             continue
81
82         custom_pointer = 0
83
84         while len(custom_chunk) > custom_pointer:
85             if not custom_pcap_skipped:
86                 custom_chunk = custom_chunk[CUSTOM_HEADER_SIZE:]
87                 custom_pcap_skipped = True
88
89             if len(custom_chunk) < custom_pointer +
                   CUSTOM_PACKET_HEADER_SIZE:
90                 break
91
```

```
92              custom_caplen = int.from_bytes(custom_chunk[
                    custom_pointer + 8:custom_pointer + 12], 'little')
93
94              if len(custom_chunk) < custom_pointer +
                    CUSTOM_PACKET_HEADER_SIZE + custom_caplen:
95                  break
96
97              custom_pointer += CUSTOM_PACKET_HEADER_SIZE +
                    custom_caplen
98
99          custom_left_overs = custom_chunk[custom_pointer:]
100
101         try:
102             if custom_chunk[:custom_pointer] != b'':
103                 custom_producer.produce(CUSTOM_KAFKA_TOPIC,
                        custom_chunk[:custom_pointer])
104         except BufferError:
105             print('Local queue full, flushing messages and trying
                    again...')
106             custom_producer.flush()
107             custom_producer.produce(CUSTOM_KAFKA_TOPIC,
                    custom_chunk[:custom_pointer])
108         finally:
109             custom_producer.poll(0)
110
111     print('Flushing final messages...')
112     custom_producer.flush()
113     custom_kafka_ended = time.time()
114
115 print(f'Custom Capture Module is running')
```

```
116 print(f'Custom Kafka Message Configuration: {CUSTOM_MESSAGE_SIZE}
        chunks')
117 print(f'Traffic been captured on the: {CUSTOM_NIC} interface')
118
119 # Start the capture as a thread
120 custom_thread_capture = threading.Thread(target=
        custom_start_writer)
121 custom_thread_capture.start()
122
123 # Let the custom_tcpdump start first
124 time.sleep(0.3)
125 custom_thread_publish = threading.Thread(target=
        custom_start_reader)
126 custom_thread_publish.start()
127
128 signal.signal(signal.SIGINT, custom_signal_handler)
129 print('Press Ctrl+C to turn off the custom capture module')
130 signal.pause()
131
132 custom_thread_capture.join()
133 custom_thread_publish.join()
134
135 # Script ended
136 custom_log_end_time = time.time()
137
138 print(
139     f'\nPackets captured: {custom_packets_captured}\nPackets
            dropped: {custom_packets_dropped}'
140     f'\nPackets received by filter: {
            custom_packets_received_by_filter}'
```

```
)
```

E.4 Training Traffic Analysis Code

```
import pandas as pd
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

scaler = MinMaxScaler()

dataset = pd.read_csv('./Train_csv_HERE.csv')

y = dataset['malicious']
x = dataset[list(
    filter(lambda x: x in ['ipvhl', 'iplen', 'ipid', 'ipfragoff',
        'iptos'
    , 'ipttl', 'ipp', 'ipsum', 'tcpdport', 'tcpsport', 'tcpseq',
       'tcpack', 'tcpoffx',
     'tcpflags', 'tcpwin', 'tcpsum', 'tcpurp', 'tcpbody'],
        dataset.columns))]

names = [
    "Random Forest"
]

models = [
```

```
    RandomForestClassifier
]

dataRest = {'modelo': [], 'accuracy': [],
            'recall': [], 'precision': [], 'f1-score': []}

X_train, X_test, y_train, y_test = train_test_split(x, y,
   test_size=0.7)

model = RandomForestClassifier()
model.fit(X_train, y_train)

filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

E.5 Training Passive DNS Analysis Code

```
import pandas as pd
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

scaler = MinMaxScaler()

dataset = pd.read_csv('./Train_CSV_HERE.csv')

y = dataset['malicious']
x = dataset[list(
```

```
14      filter(lambda x: x in ['len_domain', '
            max_len_labels_subdomain', 'number_numerical_characters',
            'ratio_numerical_characters'
15      , 'max_len_cont_num_chars', 'max_len_cont_alpha_chars', '
            max_len_cont_same_alpha_chars', 'ratio_vowels', '
            max_length_continuous_consonants', '
            number_distinct_A_records', 'ip_entropy_domain_name', '
            number_distinct_NS_records',
16       'similarity_NS_domain_name','character_entropy'], dataset.
            columns))]
17
18  names = [
19      "RBF SVM"
20  ]
21
22  models = [
23      SVC
24  ]
25
26  dataRest = {'modelo': [], 'accuracy': [],
27              'recall': [], 'precision': [], 'f1-score': []}
28
29  X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size=0.8)
30
31  model = SVC()
32  model.fit(X_train, y_train)
33
34  filename = 'finalized_model.sav'
35  pickle.dump(model, open(filename, 'wb'))
```

E.6 Complete Analysis Module Code

```
1
2  from confluent_kafka import Consumer
3  from pypacker import ppcap
4  from pypacker.layer12 import ethernet
5  from pypacker.layer3 import ip, ip6
6  from pypacker.layer4 import tcp
7  from pypacker.layer567 import http
8  import pandas as pd
9  import time
10 import csv
11 import pandas as pd
12 import pypdns
13 import re
14 import math
15 import itertools
16 import editdistance
17 from statistics import mean
18 import threading
19 import subprocess
20 import datetime
21 import signal
22 import json
23 import time
24 import os
25 import pickle
26 import tkinter as tk
27 from tkinter import messagebox
28
29
```

```
30
31 ##########CONSUMING KAFKA###################
32
33 BOOTSTRAP_SERVERS = 'kafka1.alunos.local:9093,kafka2.alunos.local
       :9093,kafka3.alunos.local:9093,kafka4.alunos.local:9093'
34
35 GROUP_ID = 'group1'
36
37 KAFKA_TOPICS = ['packetcapture']
38
39 CERTIFICATE_LOCATION = '/ssl/client.crt'
40 CERTIFICATE_KEY_LOCATION = '/ssl/client.key'
41 CA_LOCATION = '/ssl/alunos.crt'
42
43 # file location (for local storage)
44 PCAP_FILE_NAME = 'dump.pcap'
45
46 # timeout of the pool
47 POOL_WAIT = 1
48
49 CONSUMING = True
50
51 PCAP_GLOBAL_HEADER = b'\xd4\xc3\xb2\xa1\x02\x00\x04\x00\x00\x00\
       x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x01\x00\x00\x00'
52
53 # logging
54 if not os.path.exists('logs'):
55     os.mkdir('logs')
56
57 date = datetime.datetime.now()
```

```
58 LOG_FILE = f'logs/consumer_{date.year}-{date.month}-{date.day}_{
       date.hour}_{date.minute}_{date.second}.json'
59 log_start_time = time.time()
60 log_end_time = None
61 log_pcap_size_bytes = None
62 log_pcap_data_size_bytes = None
63 log_pcap_average_packet_rate = None
64 log_pcap_total_packets = None
65 log_pcap_sh1 = None
66
67 try:
68     os.remove(PCAP_FILE_NAME)
69 except FileNotFoundError:
70     pass
71
72
73 def signal_handler(sig, frame):
74     global CONSUMING
75
76     CONSUMING = False
77
78
79 def packets_consumer():
80     pcap_file = open(PCAP_FILE_NAME, 'wb')
81
82     pcap_file.write(PCAP_GLOBAL_HEADER)  # write the pcap global
           header, to get some statistics at the end
83
84     consumer = Consumer({
85         'bootstrap.servers': BOOTSTRAP_SERVERS,
```

```
86          'group.id': GROUP_ID,
87          'auto.offset.reset': 'earliest',
88          'security.protocol': 'SSL',
89          'ssl.certificate.location': CERTIFICATE_LOCATION,
90          'ssl.key.location': CERTIFICATE_KEY_LOCATION,
91          'ssl.ca.location': CA_LOCATION
92      })
93
94      consumer.subscribe(KAFKA_TOPICS)
95
96      while True:
97          message = consumer.poll(POOL_WAIT)
98
99          if not CONSUMING and message is None:
100             break
101
102         if message is None:
103             continue
104
105         pcap_file.write(message.value())
106
107     consumer.close()
108     pcap_file.close()
109
110
111 thread_consumer = threading.Thread(target=packets_consumer)
112 thread_consumer.start()
113
114 signal.signal(signal.SIGINT, signal_handler)
115 print('Press Ctrl+C to stop analysis')
```

```
116 signal.pause()
117
118 thread_consumer.join()
119
120 ###########PARSE TRAFFIC DATA####################
121
122 strtime = time.time()
123 table = pd.DataFrame()
124 aux1 = {'malicious': [],
125         'ipvhl': [],
126        'iptos': [],
127        'iplen': [],
128        'ipid': [],
129        'ipfragoff': [],
130        'ipttl': [],
131        'ipp': [],
132        'ipsum': [],
133        'tcpdport': [],
134        'tcpsport': [],
135        'tcpseq': [],
136        'tcpack': [],
137        'tcpoffx': [],
138        'tcpflags': [],
139        'tcpwin': [],
140        'tcpsum': [],
141        'tcpurp': [],
142        'tcpbody': []}
143
144 preader = ppcap.Reader(filename="traffic.pcap")
145 for ts, buf in preader:
```

```
146     eth = ethernet.Ethernet(buf)
147     if eth[ip.IP] is not None:
148         aux1["ipvhl"].append(eth[ip.IP].v_hl)
149         aux1["iptos"].append(eth[ip.IP].tos)
150         aux1["iplen"].append(eth[ip.IP].len)
151         aux1["ipid"].append(eth[ip.IP].id)
152         aux1["ipfragoff"].append(eth[ip.IP].frag_off)
153         aux1["ipttl"].append(eth[ip.IP].ttl)
154         aux1["ipp"].append(eth[ip.IP].p)
155         aux1["ipsum"].append(eth[ip.IP].sum)
156         aux1["ipsrc"].append(eth[ip.IP].src_s)
157         aux1["ipdst"].append(eth[ip.IP].dst_s)
158         if eth[tcp.TCP] is not None:
159             aux1["tcpdport"].append(eth[tcp.TCP].dport)
160             aux1["tcpsport"].append(eth[tcp.TCP].sport)
161             aux1["tcpseq"].append(eth[tcp.TCP].seq)
162             aux1["tcpack"].append(eth[tcp.TCP].ack)
163             aux1["tcpoffx"].append(eth[tcp.TCP].off_x2)
164             aux1["tcpflags"].append(eth[tcp.TCP].flags)
165             aux1["tcpwin"].append(eth[tcp.TCP].win)
166             aux1["tcpsum"].append(eth[tcp.TCP].sum)
167             aux1["tcpurp"].append(eth[tcp.TCP].urp)
168             aux1["tcpbody"].append(eth[tcp.TCP].body_bytes)
169         else:
170             aux1["tcpdport"].append("0")
171             aux1["tcpsport"].append("0")
172             aux1["tcpseq"].append("0")
173             aux1["tcpack"].append("0")
174             aux1["tcpoffx"].append("0")
175             aux1["tcpflags"].append("0")
```

```
176             aux1["tcpwin"].append("0")
177             aux1["tcpsum"].append("0")
178             aux1["tcpurp"].append("0")
179             aux1["tcpbody"].append("0")
180
181 aux1 = pd.DataFrame(aux1)
182
183 aux1.to_csv("traffic.csv")
184 endtime = time.time()
185 print("time taken: ", endtime-strtime )
186
187
188 ###########EXTRACT DNS DATA & CONSULT CIRCL DATABASE
    ###################
189 os.system("tshark -r testData.pcap -Y dns -w Data_dns.pcap")
190 lay = pyshark.FileCapture(input_file="Data_dns.pcap")
191
192 x = len([packet for packet in lay])
193
194 passive = pypdns.PyPDNS(basic_auth=('ipb.pt','
    B5fzHNadddCF4j4sXYKzNOTXYXlDZ8OXACz02wPHsk8='))
195
196 rrname = []
197 rrtype = []
198 rdata =  []
199 time_first =   []
200 time_last =    []
201 malicious = []
202
203 df = pd.read_csv("DNS.csv")
```

```
204
205 x = len(df['Domain'])
206 for i in range(x):
207
208     dm = (df['Domain'][i])
209     response = (passive.rfc_query(dm))
210     try:
211         y = len(response)
212         print(i,"=",response)
213         if response:
214             for t in range(y):
215                 response_aux = str(response[t])
216                 response_aux = response_aux.replace("PDNSRecord("
                    ,"").replace(")","")
217                 response_aux = response_aux.replace('rrname="',""
                    ).replace('rrtype="',"").replace('rdata="',"")
                    .replace("time_first=","").replace("time_last=
                    ","").replace('"',"")
218                 response_aux = response_aux.split(",")
219                 if response_aux[1] == ' A' or response_aux[1] ==
                    ' NS':
220                 #print(response_aux)
221                     rrname.append(response_aux[0])
222                     rrtype.append(response_aux[1])
223                     rdata.append(response_aux[2])
224                     time_first.append(response_aux[3])
225                     time_last.append(response_aux[4])
226     except:
227         print("\n Error \n")
228         pass
```

```
229     time.sleep(0.5)
230
231
232 info = pd.DataFrame({
233     "rrname": rrname,
234     "rrtype": rrtype,
235     "rdata":  rdata,
236     "time_first":   time_first,
237     "time_last":    time_last,
238     "malicious":    malicious
239 })
240 info.to_csv('DNS.csv')
241
242 ###########PARSE DNS DATA####################
243
244 data = pd.read_csv('DNS.csv')
245
246 def fetch_data(domain):
247     records = data[data['rrname'] == domain].to_dict('records')
248     return records
249
250 def count_distinct_A_records(domain):
251     records = data[(data['rrname'] == domain) & (data['rrtype']
        == ' A')]
252     a_records = set(records["rdata"])
253     return len(a_records)
254
255 def get_distinct_A_records(domain):
256     records = data[(data['rrname'] == domain) & (data['rrtype']
        == ' A')]
```

```
257     A_records = records["rdata"].tolist()
258     return A_records
259 
260 def calculate_ip_entropy(domain):
261     A_records = get_distinct_A_records(domain)
262     suffixes = []
263     for record in A_records:
264         regex = r"(?:\d{1,3}\.){1}\d{1,3}"
265         matches = re.finditer(regex, record, re.MULTILINE)
266         for match in matches:
267             suffixes.append(match.group())
268             break
269     unique_suffixes = set(suffixes)
270     entropy = 0
271     for n in unique_suffixes:
272         suffix_frequency = suffixes.count(n) / len(A_records)
273         entropy -= suffix_frequency * math.log2(suffix_frequency)
274     return round(entropy, 2)
275 
276 def get_distinct_NS_records(domain):
277     records = fetch_data(domain)
278     NS_records = [r['rdata'] for r in records if r['rrtype'] == '
        NS']
279     return set(NS_records)
280 
281 def retrieve_NS_records(domain):
282     ns_records = data[(data['rrname'] == domain) & (data['rrtype'
       ] == ' NS')]
283     ns_records = ns_records['rdata'].unique()
284     return ns_records
```

```
285
286 def count_distinct_NS_records(domain):
287     ns_records = retrieve_NS_records(domain)
288     return len(ns_records)
289
290 def calculate_similarity_NS_domain(domain):
291     NS_records = get_distinct_NS_records(domain)
292     results2 = []
293     for a, b in itertools.combinations(NS_records, 2):
294         results2.append(editdistance.eval(a, b))
295     if not results2:
296         return 0
297     return round(mean(results2), 2)
298
299 def length_of_domain(domain):
300     domain_no_dots = domain.replace(".", "")
301     return len(domain_no_dots)
302
303 def find_max_length_labels_subdomain(domain):
304     labels = domain.split('.')
305     max_label_length = max(len(label) for label in labels)
306     return max_label_length
307
308 def compute_character_entropy(domain):
309     all_chars = [char for char in domain.replace(".", "")]
310     distinct_chars = list(set(all_chars))
311     entropy = 0
312     for char in distinct_chars:
313         char_frequency = all_chars.count(char) / len(all_chars)
314         entropy -= char_frequency * math.log2(char_frequency)
```

```
315     return entropy
316
317 def count_numerical_characters(domain):
318     numerical_characters = re.findall(r'\d', domain)
319     return len(numerical_characters)
320
321 def calculate_ratio_numerical_characters(domain):
322     numerical_count = count_numerical_characters(domain)
323     domain_length = length_of_domain(domain)
324     if domain_length == 0:
325         return 0
326     return numerical_count / domain_length
327
328 def find_max_length_continuous_num_chars(domain):
329     labels = domain.split('.')
330     max_cont_num_chars = 0
331     for label in labels:
332         num_chars = re.findall(r'\d+', label)
333         if num_chars:
334             max_num_chars = max(num_chars, key=len, default="")
335             max_cont_num_chars = max(max_cont_num_chars, len(
                max_num_chars))
336     return max_cont_num_chars
337
338 def find_max_length_continuous_alpha_chars(domain):
339     labels = domain.split('.')
340     max_count_alpha_chars = 0
341     for label in labels:
342         alpha_chars = re.findall(r'\D+', label)
343         if alpha_chars:
```

```
344             max_alpha_chars = max(alpha_chars, key=len, default="
                    ")
345             max_count_alpha_chars = max(max_count_alpha_chars,
                    len(max_alpha_chars))
346     return max_count_alpha_chars
347
348 def find_max_length_same_alpha_chars(domain):
349     domain = ''.join(filter(str.isalpha, domain))
350     max_same_alpha_chars = 0
351     current_count = 1
352     for i in range(len(domain) - 1):
353         if domain[i] == domain[i + 1]:
354             current_count += 1
355         else:
356             max_same_alpha_chars = max(max_same_alpha_chars,
                    current_count)
357             current_count = 1
358     max_same_alpha_chars = max(max_same_alpha_chars,
           current_count)
359     return max_same_alpha_chars
360
361 def calculate_ratio_vowels(domain):
362     vowels = "AEIOUaeiou"
363     vowels_count = sum(domain.count(v) for v in vowels)
364     domain_length = length_of_domain(domain)
365     if domain_length == 0:
366         return 0
367     return vowels_count / domain_length
368
369 def find_max_length_continuous_consonants(domain):
```

```
370     labels = domain.split('.')
371     max_cont_consonants = 0
372     for label in labels:
373         consonant_chars = re.findall(r'[b-df-hj-np-tv-z]+', label
                , re.IGNORECASE)
374         if consonant_chars:
375             max_consonant_chars = max(consonant_chars, key=len,
                    default="")
376             max_cont_consonants = max(max_cont_consonants, len(
                    max_consonant_chars))
377     return max_cont_consonants
378
379 unique_domains = data['rrname'].unique()
380
381 results = {
382     "Domain": unique_domains,
383     "Length of Domain": [],
384     "Max Length of Labels Subdomain": [],
385     "Character Entropy": [],
386     "Number of Numerical Characters": [],
387     "Ratio of Numerical Characters": [],
388     "Max Length of Continuous Numerical Characters": [],
389     "Max Length of Continuous Alpha Characters": [],
390     "Max Length of Same Alpha Characters": [],
391     "Ratio of Vowels": [],
392     "Max Length of Continuous Consonants": [],
393     "Number of Distinct A Records": [],
394     "IP Entropy of Domain Name": [],
395     "Number of Distinct NS Records": [],
396     "Similarity of NS Domain Name": []
```

```
}

for domain in unique_domains:
    results["Length of Domain"].append(length_of_domain(domain))
    results["Max Length of Labels Subdomain"].append(
        find_max_length_labels_subdomain(domain))
    results["Character Entropy"].append(compute_character_entropy
        (domain))
    results["Number of Numerical Characters"].append(
        count_numerical_characters(domain))
    results["Ratio of Numerical Characters"].append(
        calculate_ratio_numerical_characters(domain))
    results["Max Length of Continuous Numerical Characters"].
        append(find_max_length_continuous_num_chars(domain))
    results["Max Length of Continuous Alpha Characters"].append(
        find_max_length_continuous_alpha_chars(domain))
    results["Max Length of Same Alpha Characters"].append(
        find_max_length_same_alpha_chars(domain))
    results["Ratio of Vowels"].append(calculate_ratio_vowels(
        domain))
    results["Max Length of Continuous Consonants"].append(
        find_max_length_continuous_consonants(domain))
    results["Number of Distinct A Records"].append(
        count_distinct_A_records(domain))
    results["IP Entropy of Domain Name"].append(
        calculate_ip_entropy(domain))
    results["Number of Distinct NS Records"].append(
        count_distinct_NS_records(domain))
    results["Similarity of NS Domain Name"].append(
        calculate_similarity_NS_domain(domain))
```

```
414
415     records = data[(data['rrname'] == domain)]
416     print(domain)
417
418 final_data = pd.DataFrame(results)
419 final_data.to_csv("FinalDNS.csv", index=False)
420
421
422
423 ###########ANALYSIS TRAFFIC MODULE###################
424
425 model = pickle.load(open('finalized_model.sav', 'rb'))
426 dataset = pd.read_csv('./testtable.csv')
427 rows = len(dataset)
428
429 x = dataset[list(
430     filter(lambda x: x in ['ipvhl', 'iplen', 'ipid', 'ipfragoff',
            'iptos'
431     , 'ipttl', 'ipp', 'ipsum', 'tcpdport', 'tcpsport', 'tcpseq',
           'tcpack', 'tcpoffx',
432      'tcpflags', 'tcpwin', 'tcpsum', 'tcpurp', 'tcpbody'],
            dataset.columns))]
433
434 print("Receveid ",rows," packets from the capture\n")
435
436 startAnalysis = time.time()
437 print("Starting analyzing at: ", startAnalysis )
438 predicted = model.predict(x)
439
440 i = 0
```

```
441 hosts = dataset['ipsrc']
442 malicious = []
443 while(i < rows):
444     if(predicted[i] == 1):
445         malicious.append(hosts[i])
446     i = i+1
447 
448 unique_domains = list(dict.fromkeys(malicious))
449 nDomain = len(unique_domains)
450 i=0;
451 
452 while(i<nDomain):
453     mess = "Malicious Activity coming from host: ",
        unique_domains[i]
454     messagebox.showwarning(title="MALICIOUS ACTIVITY FOUND",
        message=mess)
455     i = i+1
456 
457 end = time.time()
458 print("Finish analysis at: ", end-startAnalysis)
459 
460 
461 ##########ANALYSIS DNS MODULE###################
462 
463 model = pickle.load(open('finalized_model_DNS.sav', 'rb'))
464 dataset = pd.read_csv('./FinalDNS.csv')
465 rows = len(dataset)
466 
467 x = dataset[list(
```

```
468     filter(lambda x: x in ['len_domain', '
            max_len_labels_subdomain', 'number_numerical_characters',
            'ratio_numerical_characters'
469     , 'max_len_cont_num_chars', 'max_len_cont_alpha_chars', '
            max_len_cont_same_alpha_chars', 'ratio_vowels', '
            max_length_continuous_consonants', '
            number_distinct_A_records', 'ip_entropy_domain_name', '
            number_distinct_NS_records',
470      'similarity_NS_domain_name','character_entropy'], dataset.
            columns))]
471
472 print("Receveid ",rows," packets from the capture\n")
473
474 startAnalysis = time.time()
475 print("Starting analyzing at: ", startAnalysis )
476 predicted = model.predict(x)
477 end = time.time()
478 print("Finish analysis at: ", end-startAnalysis)
```