REAL-TIME INTRUSION DETECTION AND PREVENTION SYSTEM FOR SDN-BASED IOT NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALPER KAAN SARIÇA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

REAL-TIME INTRUSION DETECTION AND PREVENTION SYSTEM FOR SDN-BASED IOT NETWORKS

submitted by **ALPER KAAN SARIÇA** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ertan Onur
Computer Engineering, METU _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University _____

Date: 02.09.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Alper Kaan Sarıça

Signature :

ABSTRACT

REAL-TIME INTRUSION DETECTION AND PREVENTION SYSTEM FOR SDN-BASED IOT NETWORKS

Sarıça, Alper Kaan
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Pelin Angın

September 2021, 68 pages

The significant advances in wireless networks with the 5G networks have made possible a variety of new IoT use cases. 5G and beyond networks will significantly rely on network virtualization technologies such as SDN and NFV. The prevalence of IoT and the large attack surface it has created calls for SDN-based intelligent security solutions that achieve real-time, automated intrusion detection and mitigation. In this thesis, we propose a real-time intrusion detection and mitigation system for SDN, which aims to provide autonomous security in the IoT networks. The proposed approach is built upon automated flow feature extraction and classification of flows using random forest classifier at the SDN application layer. We present an SDN-specific dataset we generated for IoT and provide performance of the proposed intrusion detection model. In addition to the model performances, we provide network experiment results in the presence and absence of our proposed security mechanism. Experiment results demonstrate that the proposed security approach is promising to achieve real-time, highly accurate detection and mitigation of attacks in SDN-managed IoT networks.

Keywords: SDN, 5G, security, machine learning, IoT, intrusion detection

ÖZ

NESNELERİN İNTERNETİ İÇEREN YAZILIM TANIMLI AĞLARDA GERÇEK ZAMANLI SALDIRI TESPİTİ VE ÖNLENMESİ SİSTEMİ

Sarıça, Alper Kaan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Pelin Angın

Eylül 2021 , 68 sayfa

5G ağlarıyla beraber kablosuz ağlardaki gelişmeler birçok yeni nesnelerin interneti senaryosunu mümkün kıldı. 5G ve ötesi ağları network sanallaştırması için yazılım tanımlı ağları ve ağ fonksiyonlarını sanallaştırmayı kullanacak. Nesnelerin internetinin yaygın kullanımı ve güvenlik açıkları yazılım tanımlı ağ tabanlı akıllı, gerçek zamanlı ve otonom saldırı tespit ve önlenmesi sistemlerini gerekli kılıyor. Bu tezde, yazılım tanımlı ağ tarafından yönetilen nesnelerin interneti ağları için gerçek zamanlı ve otonom bir saldırı tespiti ve önlenmesi sistemi öneriyoruz. Önerilen yaklaşım yazılım tanımlı ağların uygulama katmanında rastgele orman modeli kullanarak otomatik bir şekilde akışlardan özellik elde etmek ve bunları sınıflandırmak üzerine kurulu. Nesnelerin interneti için yazılım tanımlı ağlara özel dataset ürettik ve önerdiğimiz saldırı tespit modelimizin performansı gösterdik. Model performanslarına ek olarak, önerdiğimiz sistemin varlığındaki ve yokluğundaki ağ deneyleri sonuçlarını paylaştık. Deney sonuçlarının gösterdiği gibi önerdiğimiz güvenlik yaklaşımı yazılım tanımlı ağ tarafından yönetilen nesnelerin interneti ağları için gerçek zamanlı ve yüksek doğrulukla saldırıları tespit ediyor ve önlüyor.

Anahtar Kelimeler: yazılım tanımlı ağ, 5G, güvenlik, makine öğrenmesi, nesnelerin interneti, saldırı tespiti

To my family

ACKNOWLEDGMENTS

First of all, I would like to thank my thesis advisor Assist. Prof. Dr. Pelin Angın for her guidance, patience and continuous support. Without her help, it would not be possible to complete this thesis.

I would like to thank Prof. Dr. Ertan Onur for the opportunities and guidance he provided.

I would like to thank Assoc. Prof. Dr. Ahmet Burak Can for his feedback.

I would like to thank Türk Telekom and BTK for supporting this thesis under 5G and Beyond Joint Graduate Program. I also wish to thank Selami Çiftçi, my advisor at Türk Telekom, for his help and suggestions.

I would like to thank TUBİTAK for the scholarship (BİDEB 2210-A).

I would like to thank the members of the WINS Lab for the great study environment.

Lastly, I would like to express my deepest gratitude to my brother and parents for being always there for me no matter the odds.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

DoS	Denial of Service
DDoS	Distributed Denial of Service
IoT	Internet of Things
ML	Machine learning
NFV	Network function virtualization
OS	Operating system
QoS	Quality of Service
RF	Random forest
SDN	Software-defined Networking

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

The number of connected devices and Internet of Things (IoT) use cases have been continuously increasing thanks to the developments in the fields of mobile networks, big data and cloud computing. IoT use cases significantly facilitating our daily lives include smart homes, autonomous cars, security systems, smart cities and remote healthcare among many others. When the large volumes of data generated by IoT is considered, it is obvious that quality of service (QoS) requirements of these various use cases will not be satisfiable by legacy wireless networks. 5G and beyond networks relying on software-defined networking (SDN) and network function virtualization (NFV) for resource management will be a key enabler for the future's ubiquitous IoT.

IoT has already resulted in a large attack surface due to limited processing power and battery life as well as the lack of security standards, which make a large number of IoT devices incapable of implementing even basic security mechanisms like encryption. New use cases, protocols and technologies add new attack surfaces to the existing ones. It is of utmost importance to develop intrusion detection and prevention systems for IoT networks that address new and existing vulnerabilities in order to ensure healthy operation of these systems. It is also essential to ensure compliance of the developed security techniques with SDN-based network architectures and benefit from the network programmability provided by SDN to ensure fast detection and mitigation of attacks, as well as quick reconfiguration of the networks to prevent QoS degradation and failures.

Machine learning (ML) techniques have become popular tools for network intrusion

detection tasks in the past two decades, especially due to the increasing accuracy achieved by a variety of models, and the superiority they have over rule-based systems in detecting previously unseen attacks. The developments in the field of deep learning have made them indispensable parts of any classification task, including intrusion detection. Although deep learning models have been shown to be quite successful in intrusion detection, they are usually used as blackboxes, and their decision-making processes are not readily explainable to human experts [1]. The explainability problem is especially important in the security domain [2] to correctly interpret the results produced by these models.

Despite the importance of realistic network traffic data for effective model building, most of the existing research in IoT intrusion detection has used datasets that were generated for legacy networks without IoT traffic. This is mostly due to the lack of publicly available datasets that include IoT traffic, except the recently released Bot-IoT dataset [3]. To the best of our knowledge, there is no publicy available dataset specifically for SDN-based IoT environments. The network traffic characteristics of IoT and SDN are quite different from those of legacy networks, therefore, using models trained with legacy network data might lead to inaccurate classification results. Furthermore, it is crucial for intrusion detection systems to retrieve features in real time for effective attack detection and mitigation. Existing public datasets have been created by processing pcap files and there is no guarantee that all of the features they include can be retrieved in real time.

1.2 Contributions and Novelties

In an effort to address the abovementioned shortcomings of existing security approaches for SDN-based IoT networks, in this thesis we introduce a real-time intrusion detection and prevention system.

Our contributions are summarized as follows:

- We propose an end-to-end intrusion detection and prevention system architecture for SDN-based networks, which monitors traffic flows at the controller and classifies the traffic using an intelligent attack detection module based on

machine learning, according to which it makes flow rule updates.

- We generate a novel SDN dataset in an IoT network including normal traffic and different types of attack traffic.
- We evaluate the performance of the proposed dataset with different machine learning algorithms and identify the most important features.
- We provide network performance results in the presence and absence of our proposed security system and show that our system achieve real time protection.

1.3 The Outline of the Thesis

The rest of this thesis is organized as follows:

Chapter 2 provides a brief background on SDN and random forest classifier. Chapter 3 reviews related work in techniques for intrusion detection in SDN-based networks and existing ML datasets for network intrusion detection. Chapter 4 describes our proposed end-to-end intrusion detection and prevention approach for SDN-based networks. Chapter 5 provides details of the generated SDN datasets and classification results. Chapter 6 provides experimental evaluation of the proposed intrusion detection and prevention system. Chapter 7 concludes the thesis with future work directions.

CHAPTER 2

BACKGROUND

This chapter provides an overview of software-defined networking (SDN) and the random forest classifier, which are key components of the proposed solution.

2.1 Software-Defined Networking (SDN)

Software-defined networking (SDN) emerged as a novel networking paradigm in the past decade, supporting the need for programmatically managing networks, the operational costs of which were increasing sharply with the widespread use and new technologies that are needed to accommodate various IoT use cases. SDN differs from traditional networks by separating the data and control planes, where routers/switches are now responsible for forwarding functionality, where routing decisions are taken by the controller (control plane).

The SDN architecture mainly consists of three layers: applications, control, and infrastructure (data plane), as seen in Figure 2.1. All of the applications, such as load balancing and intrusion detection systems, run on the application layer and communication with the controller takes place through the north-bound API. Communication between the controller and switches takes place through the south-bound API, mainly using the OpenFlow [4] protocol. The logically centralized controller is responsible for managing the network. The controller maintains a global view of the network and installs forwarding rules, called "flow rules", into the corresponding switches based on the routing decisions it makes. Switches store flow rules in their flow tables and forward network packets based on matches with existing flow rules.

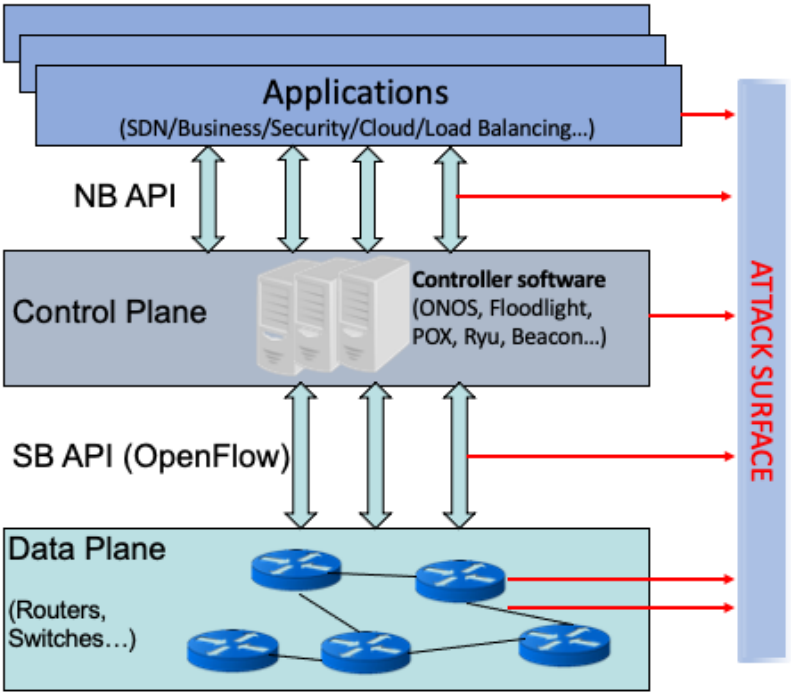

Figure 2.1: Software-defined networking (SDN) Architecture and Attack Surface.

Figure 2.2 shows the structure of a flow rule. It is mainly composed of three parts: match fields, counters, and actions. Unlike traditional networks that perform forwarding based on the destination addresses, match fields are determined by the configuration of the forwarding application and might be ingress port, VLAN ID, source and/or destination MAC addresses, IP addresses, and/or port numbers. Counters keep track of the duration of the flow and byte and packet counts that matched the flow. The action can be forwarding the packet to the specified port or dropping it, among others.

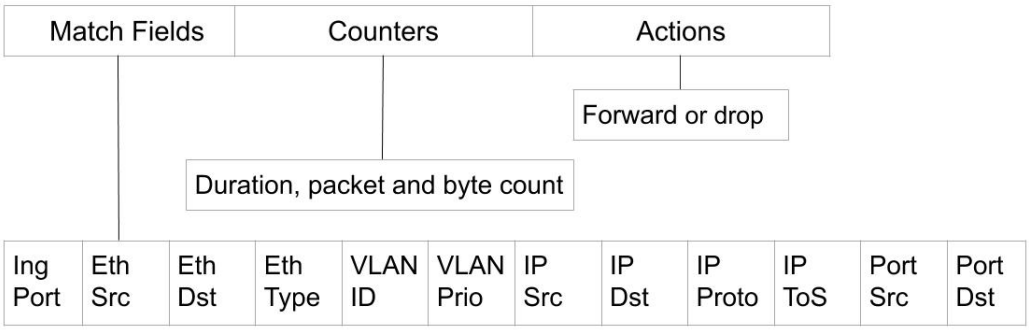

Figure 2.2: Flow Rule Structure.

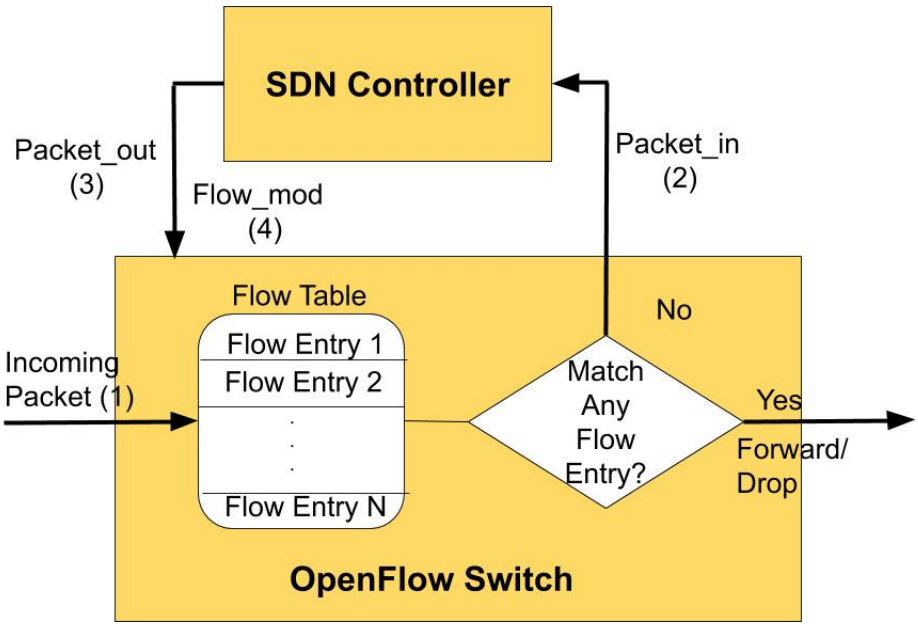

Figure 2.3: Packet Flow in SDN.

Figure 2.3 illustrates operations performed on the incoming packets in SDN. The header fields of incoming packets are compared with the match fields of flow rules in the switch. All of the required header fields of the packet should match with the match fields of a rule in the flow table to be forwarded immediately. Otherwise, the switch will buffer the packet and send what is called a *packet_in* message to the controller that contains the header fields of the packet. The controller then examines the packet_in message and generates a routing decision for the packet, which is sent back to the switch in a *packet_out* message. The necessary action is taken for the packet and the corresponding flow rules are installed to the switches by sending *flow_mod* messages. Even though the controller decides to install flow rules into the corresponding switches, a packet_out message is always sent before the flow_mod message. Therefore, unlike traditional networks, the statistics of the first packet that triggered flow rule installation cannot be seen in the installed flow rule. For TCP connections, statistics of the SYN and SYN ACK packets are lost, because they are the first packets sent from source to destination and destination to source, respectively.

Due to the packet matching and flow rule installation behaviors of SDN, DoS and DDoS attacks also affect the SDN controller. During DoS and DDoS attacks with spoofed addresses, all of the incoming packets may have different source addresses.

Therefore, all of the incoming packets from the attacker may trigger a new flow rule installation. Packet_in requests caused by the DoS and DDoS attacks decrease the available bandwidth between the controller and the switch. These packet_in requests also consume the resources of the SDN controller, i.e. CPU, memory.

2.1.1 SDN in 5G Networks

While early adoptions of SDN mostly took place in wired enterprise networks, its flexibility, programmability, speed, and cost advantages have recently made it a promising tool for other networks, including wireless sensor networks (WSNs) [5] and next generation wireless networking infrastructures. SDN will be one of the greatest enablers of 5G and beyond networks by providing the network virtualization capabilities that are needed to remotely and dynamically manage the networks. The fast failover and autonomous management capabilities to be achieved with SDN applications will provide the high bandwidth and low delay requirements of 5G networks, making them support a variety of IoT use cases. SDN, together with network functions virtualization (NFV), will especially form the basis of network slicing in 5G core and radio access networks, which will be a significant enabler for operators to efficiently utilize their infrastructure in order to provide the required quality of service and security guarantees to their customers [6].

A number of SDN-based architectures for 5G networks have been proposed [7]. One of the early proposals is SoftAir by Akyildiz et al. [8], where the data plane is a programmable network forwarding infrastructure that consists of software-defined core network (SD-CN) and software-defined radio access network (SD-RAN). While SD-RAN contains software-defined base stations, including small cells (microcells, femtocells, and picocells) in addition to traditional macro cells, SD-CN contains software-defined switches that form the 5G core network, as seen in Figure 2.4. User equipment and other devices are connected to the software-defined base stations or wireless access points, which are connected to the software-defined core network through the backhaul links. As proposed in SoftAir, SD-CN and SD-RAN can both use OpenFlow as the southbound API, which will provide a uniform interface with the controller routing traffic from the base stations through the optimal paths in the core

network. This architecture enables the application of many of the same principles in terms of network control from wired SDN to SDN-based 5G networks.

One of the biggest promises of and reasons for the introduction of SDN is the provisioning of improved security in the network through global visibility, and the fast automated reconfiguration of flow rules. This will enable real-time detection and mitigation of malicious traffic in the network. As seen in Figure 2.1, an SDN can be attacked at various surfaces (the attack surface is demonstrated by red arrows pointing out from the devices, applications, or interfaces that could be attacked in SDN). These attacks could not only target the data plane devices, but also the controller and applications to cause disruptions in network operation. In this work, we focus on attacks that affect the data and control planes and propose an intrusion detection and mitigation solution that provides automated responses to attacks detected while using highly interpretable ML algorithms that are described in the next section.

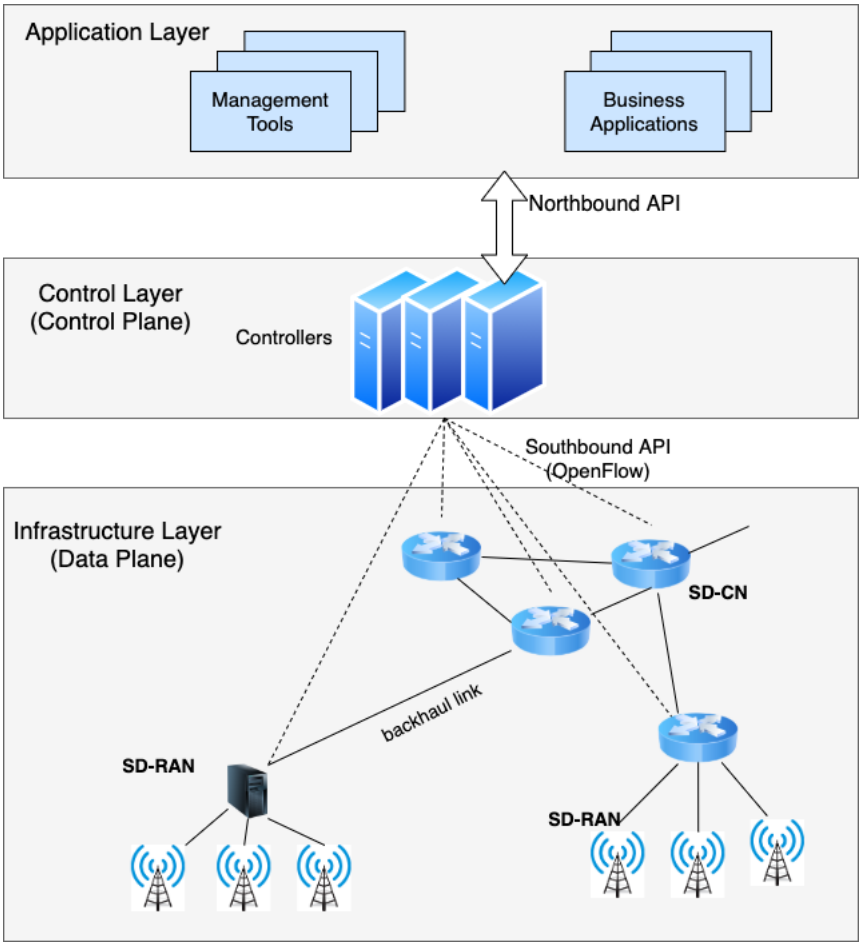

Figure 2.4: Software-Defined 5G Network Architecture.

2.2 Random Forest Classifier

Random forest (RF) is a machine learning model that constructs an ensemble of decision trees, named a forest, such that each decision tree is constructed using an independently and identically distributed random vector [9]. For classifying a particular data instance, a random forest uses the outputs of all trees in the forest to pick the majority decision. The utilization of the outputs of multiple trees makes the classifier more robust than decision trees, which suffers from the overfitting problem in many cases.

At a high level, the RF algorithm works as follows:

1. The complete training set S consisting of n data instances with class labels $\{c_i$, i = 1, ..., n$\}$ from a set of classes C is split into K random subsets using bootstrap sampling:

$$S = S_1, S_2, ..., S_K \tag{2.1}$$

2. A random feature vector θ_k is created and used to build a decision tree from each S_k. All $\{\theta_k$, k=1, 2, 3, ..., $\theta_K\}$ are independent and identically distributed.

3. Each tree r(S_k, θ_k) is grown without pruning to form the forest R.

4. The classification of a test data instance x is calculated as follows:

$$H(x) = max_{C_j} \sum_{i=1}^{k} (I(h_i(x) = C_j)) \tag{2.2}$$

   where I is the indicator function, and $h_i(x)$ is the result of classification by $r(S_i, \theta_i)$.

Figure 2.5 shows a simplified view of classification by random forests. A commonly used metric for deciding the splitting criteria for the various nodes in the decision trees is information gain. The information gain from the split of a node S based on a random variable a is calculated as follows:

$$IG(S, a) = E(S) - E(S|a) \tag{2.3}$$

Here, $E(S)$ is the entropy of the parent node before the split and E(S|a) is the weighted average of the entropies of the child nodes after the split. $E(S)$ is calculated as:

$$E(S) = -\sum_{i=1}^{C} p(c_i) log p(c_i) \tag{2.4}$$

where $p(c_i)$ is the probability of data instance in node S having class label c_i.

Figure 2.6 shows a partial view of a decision tree from the random forest constructed for a sample network intrusion detection task on the IoT network dataset we have generated. As seen in the figure, the entropy of nodes decreases while approaching the leaves, as nodes higher up in the tree are split based on a specific threshold of feature values as discovered by the algorithm. A random forest contains a multitude of such decision trees, each constructed from a different, randomly sampled subset of the whole training data.

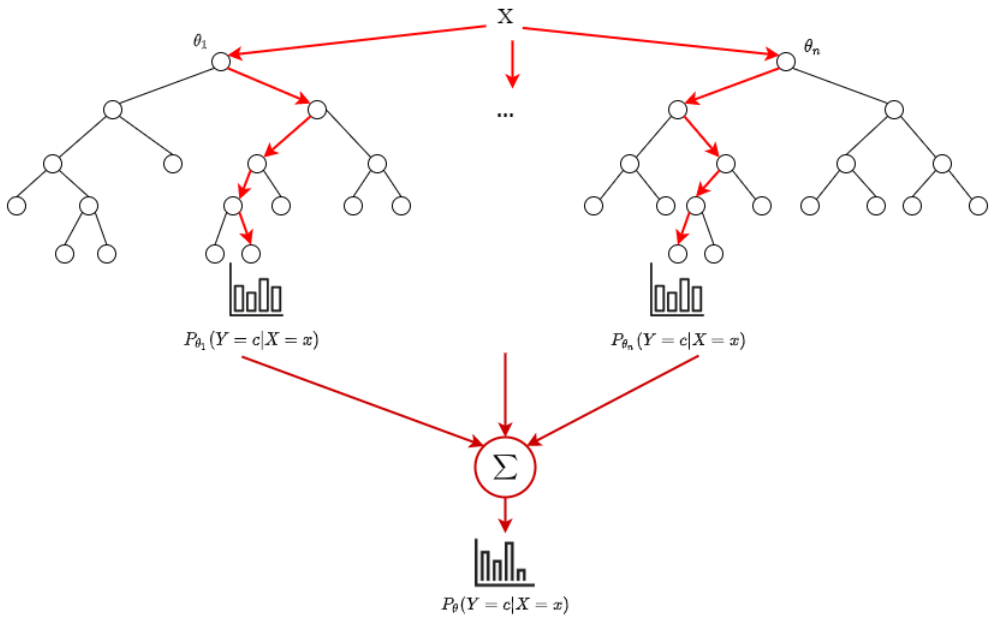

Figure 2.5: Classification by random forest.

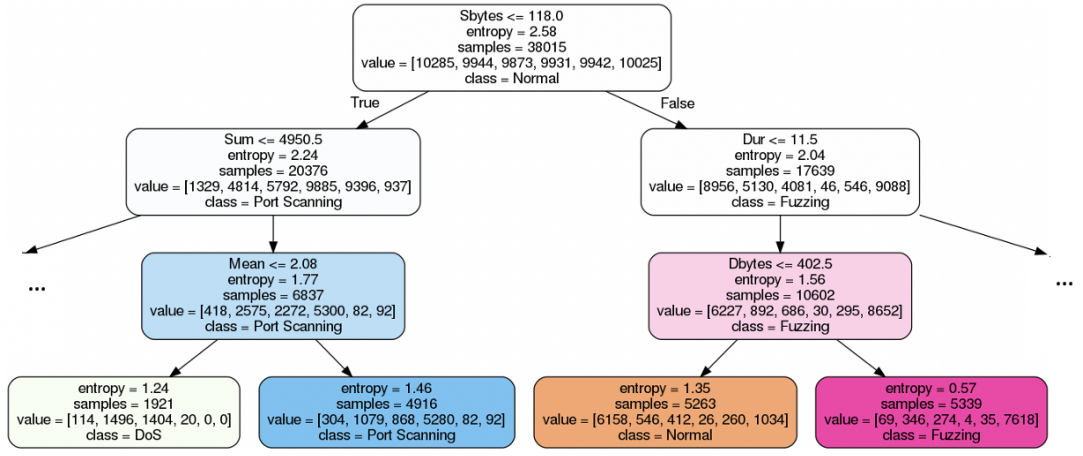

Figure 2.6: Sample decision tree from random forest for a network intrusion detection task.

RF is among the ML algorithms with the highest degree of explainability/interpretability due to its reliance on decision trees, which construct models based on splits of training data along feature values, easily readable by human domain experts. The effectiveness of RF for a variety of classification tasks has been shown in many studies [10]. Despite the success of especially deep learning algorithms in various classification tasks in recent years, RF continues to outperform many state-of-the-art ML algorithms, especially in tasks involving structured data.

CHAPTER 3

RELATED WORK

Intrusion detection and mitigation in networks has become an ever more important topic of research with the increasing cyber security incidents, caused by the large attack surfaces created by IoT. With the increasing adoption of SDN-based network architectures in the past decade, SDN security has become one of the centers of attention for the cyber security research community.

Rathore and Park [11] proposed a fog-based semi-supervised learning approach for distributed attack detection in IoT networks. The authors used NSL-KDD dataset and showed their distributed approach performed better than centralized solution in terms of detection time and accuracy. Evmorfos et. al. [12] proposed an architecture that uses Random Neural Network and LSTM to detect SYN flooding attacks in IoT networks. The authors generated their dataset by creating a virtual network and recorded the traffic into pcap files. Soe et. al. [13] proposed a sequential attack detection architecture that uses three machine learning models for IoT networks. The authors used N-BaIoT dataset and achieved 99% accuracy. Alqahtani et. al. [14] proposed a genetic-based extreme gradient boosting (GXGBoost) model that uses Fisher-score to select features in IoT networks. The Authors also used N-BaIoT dataset and achieved 99.96% accuracy. Even though these approaches successfully detects attacks against IoT, they work in traditional networks.

The majority of the solutions proposed for SDN-based networks so far have focused on techniques for detection and mitigation of denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks. In [15] a semi-supervised model on the UNB-ISCX dataset was used to detect DDoS attacks in SDN-based IoT networks. The model achieved above 96% accuracy on the UNB-ISCX dataset and the own dataset of the

authors, which included only UDP flooding attacks. In [16] an entropy-based solution was proposed for detection and mitigation of DoS and DDoS attacks in software-defined IoT networks. The approach achieved high accuracy on the Bot-IoT dataset and the authors' dataset containing TCP SYN flooding attacks. Yin et. al. [17] proposed to use cosine similarity of packet_in rates received by the controller and drop packets if a predefined threshold is reached. Their approach only mitigated DDoS attacks. Ahmed and Kim [18] proposed an inter-domain information exchange approach that uses statistics collected from switches across different domains to mitigate DDoS attacks. Bhunia and Gurusamy [19] used Support Vector Machine (SVM) to detect and mitigate DoS attacks in SDN-based IoT networks. The authors created their own data, however the dataset is not publicly available. Sharma et. al. [20] proposed to use deep belief networks to mitigate DDoS attacks in SDN-based cloud IoT. Bull et. al. [21] used an SDN gateway to detect and block anomalous flows in IoT networks. Their approach managed to succesfully detect and mitigate TCP and ICMP flooding attacks. In spite of the fact that most of these approaches have accomplished successful detection and mitigation, they only work against DoS and DDoS attacks.

Other works have targeted coverage of additional attacks, but used datasets not specific to SDN for evaluation. Li et al. [22] proposed to use the BAT algorithm for feature selection and then used the random forest algorithm on the KDD CUP'99 dataset, achieving 96% accuracy. In [23] the CART decision tree algorithm was proposed to detect anomalies in IoT networks using SDN. The authors used the CICIDS'2017 dataset and achieved 99% detection rate. Dawoud et. al. [24] proposed an SDN-based framework for IoT that uses Restricted Boltzmann Machines to detect attacks. Authors achieved higher detection rate than existing works on KDD CUP'99 dataset. Al Hayajneh et al. [25] proposed a solution for detecting man in the middle attacks against IoT in SDN. Their solution works only for IoT devices that use HTTP for communication. Shafi et. al. [26] proposed a fog-assisted SDN-based intrusion detection system for IoT that uses Alternate Decision Tree. The authors used the UNSW-NB15 dataset and achieved high detection rates. Derhab et. al. [27] proposed an intrusion detection system which uses Random Subspace Learning, K-Nearest Neighbor and blockchain against attacks which target industrial control processes. The authors used Industrial Control System Cyber attack dataset and demonstrated their solution

achieves high accuracy. Work in explainable intrusion detection systems have been rather limited so far. Wang et al. proposed an explainable machine learning framework for intrusion detection systems based on Shapley Additive Explanations [28]. The framework was evaluated on the NSL-KDD dataset.

Some attack types also affect the SDN controller and switches. In [29], authors measured the available bandwidth between the SDN controller and the switches during DoS attack. Alharbi et. al. [30] showed the effects of DoS attacks in SDN. Authors measured the packet delivery ratio, controller cpu utilization and switch cpu utilization under DoS attack. Mousavi and St-Hilaire [31] used entropy of the destination IP addresses to detect DDoS attacks againts SDN controller and achieved 96% detection rate. Gkountis et al. [32] proposed a rule based approach to protect the SDN controller against DDoS attacks. Simsek et. al. [33] showed the effects of different types of DoS attacks on the SDN and proposed an approach to mitigate DoS attacks in the data plane using P4 switches.

Network intrusion detection using ML techniques has been a popular approach of network security especially for the past two decades, for which researchers have created a number of extensive network trace datasets. These datasets, even if old, are still in use today by security researchers as benchmarks. Among existing publicly available network intrusion detection datasets are the following:

- **KDD CUP'99** dataset [34] was generated in 1999 by extracting features from the DARPA98 [35] dataset, which simulates a U.S. Air Force LAN. KDD CUP'99 has 41 features and 4 attack categories: DoS, R2L, U2R and probing. Even though it is an old dataset, many researchers still use this dataset. However, it is not without some drawbacks. Firstly, the distribution of the records in the training and test sets are widely different, because the test set includes some attack types that are not in the training set [36]. Secondly, around 75% of the data in the training and test sets are duplicates, which could lead to biased classification models. Most importantly, the dataset was not generated in an IoT environment and does not include SDN-specific features.

- **NSL-KDD** dataset [36] was created by improving KDD CUP'99 dataset. The aim of this dataset was to solve problems of KDD CUP'99 dataset. Duplicate

records were eliminated and number of records were reduced. Also classes were balanced. Still, this dataset does not represent the behavior of current networks.

- **UNB-ISCX** dataset [37] was created by the Canadian Institute of Cybersecurity in 2012. Real network traces were analyzed to create realistic profiles. The dataset consist of 7 days of network traffic containing three types of attacks: DDoS, brute force SSH and infiltrating the network from inside.

- **CAIDA** dataset [38] contains anonymized network traces. Records were created by removing the payloads of the packets and anonymizing the header. This dataset only contains DoS attacks and features are the header fields. Additional features using the header field were not generated.

- **UNSW-NB15** dataset [39] was created in 2015. IXIA tool were used to generate the network traffic. UNSW-NB15 dataset has 49 features and two of them are labels for binary and multi-class classification. The dataset consists of normal traffic and 9 types of attack traffic, namely DoS, DDoS, fuzzing, backdoor, analysis, exploit, generic, worm and shellcode. The main problem of the dataset is the lack of enough number of samples for some attack types.

- **CICIDS2017** dataset [40] is another dataset created by the Canadian Institute of Cybersecurity. Realistic benign traffic were created using their B-Profile system. The dataset includes normal traffic and 6 types of attack traffic, namely DoS, botnet, port scannig, brute force, infiltration an web attack.

- **Bot-IoT** dataset [3] was introduced in 2018. The most important part of the dataset is that it includes IoT traffic, unlike most of the existing intrusion detection datasets. The dataset has 46 features and two of them are labels for binary and multi-class classification. The dataset consist of normal traffic and 6 different attack types, namely DoS, DDoS, service scanning, OS fingerprinting, data theft and keylogging. The main problem of the dataset is the lack of enough number of samples for some attack types. Number of records for normal traffic is also low.

Most of the existing work on intrusion detection systems for IoT and SDN envi-

ronments used the datasets mentioned above. However, none of these datasets have been created in an SDN environment, therefore they do not reflect SDN network behaviours. Furthermore, these datasets do not contain IoT traffic, except the BoT-IoT dataset. Most of the existing datasets were created by recording and processing pcap files with different tools. Therefore, an SDN controller may not be able to get all of the features in real time. To the best of our knowledge, there is no other publicly available SDN dataset which includes IoT traffic.

CHAPTER 4

PROPOSED SECURITY APPROACH

The proposed intrusion detection and mitigation approach, the overall operation of which is depicted in Figure 4.1, provides security in SDN-based networks by automated, intelligent analysis of network flows, followed by mitigation actions taken in accordance with the decision of the intrusion detection component. The end-to-end intrusion detection and mitigation process relies on three main modules running on an SDN application in the application layer, namely Feature Creation, RF classifier and Attack Mitigation.

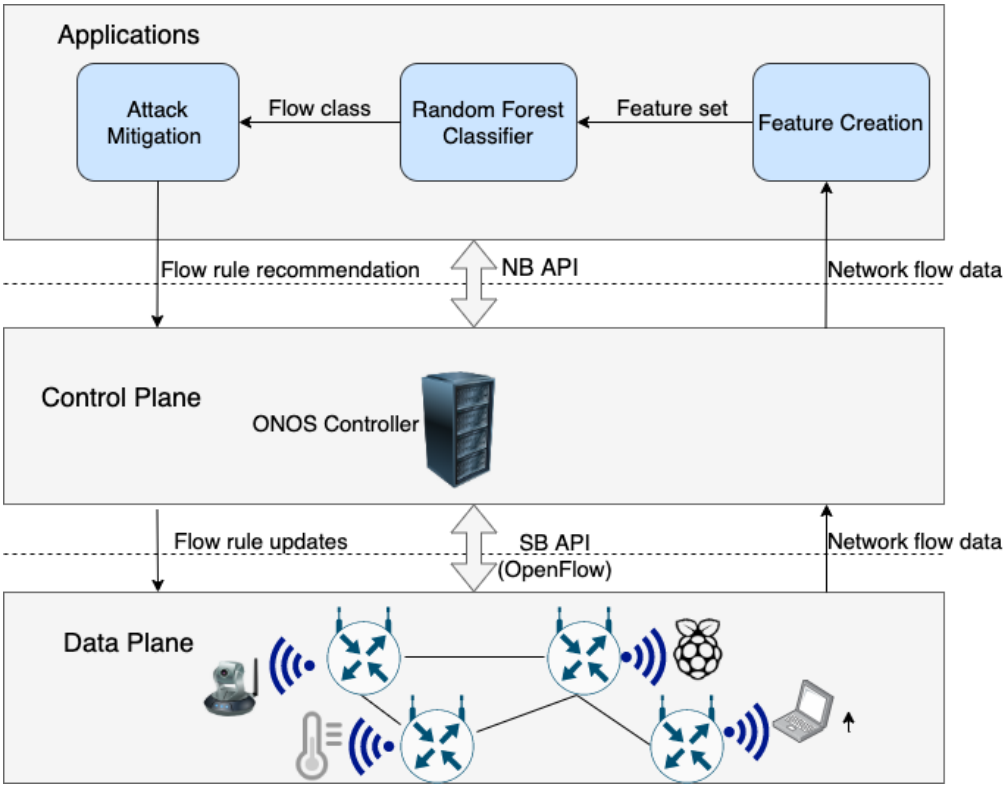

Figure 4.1: SDN-based Security Solution Architecture.

The Feature Creation module collects network flows from the switches at regular intervals and calculates features required by the RF classifier for each flow. RF classifier applies its pre-built intrusion detection model on the flow instance and passes the result to the Attack Mitigation module. The Attack Mitigation module then determines the action to take based on the classification result and installs flow rules into the corresponding switches to mitigate the attack if necessary. Algorithm 1 summarizes the end-to-end operation of the proposed security solution.

The controller periodically collects network flow entries from the switches, which are retrieved by the Feature Creation module at regular intervals. Upon retrieval, features are created for every flow. Common features for every flow are generated looping over every flow entry in the switch. E.g. average duration of flows and total number of packets in transaction, are created with an initial pass over flow entries. Then, flow-specific features, e.g. duration of flow and source-to-destination packet count, are retrieved by passing over all of the flow entries. While looping over flow entries, the created feature vector for a flow is sent immediately to the RF classifier, without waiting to finish feature creation for other flow entries. The Feature Creation module also retrieves flow match fields like source IP and MAC addresses and physical port of the switch where the packet is coming from. These features are used by the Attack Mitigation module. Details of the algorithm for the creation of features used by our model will be provided in the Chapter 6.

The RF classifier, which works as explained in Chapter 2 gets feature vectors from the Feature Creation module one by one and classifies them using its pre-built intrusion detection model. If the outcome of the classification is any attack type, detected attack type and source identifiers of the flow, i.e. source IP, source MAC address and physical switch port the packet is coming from, are sent to the Attack Mitigation module. The used machine learning model should be updated dynamically by inclusion of new training data for existing attack types or adding new attack types as they are discovered. The RF model built is multi-class classification model formed using training data consisting of various attack types in addition to normal traffic. We advocate using multi-class attack classification rather than binary classification/anomaly detection, as the former provides more informed decision-making capability in terms of the action to take/the specific flow rule to install.

$L_S \leftarrow$ Get connected SDN switches

$M \leftarrow$ Load RF classifier model

$L_B \leftarrow$ Blacklist

while *True* **do**

  forall S *in* L_S **do**

    $L_E \leftarrow$ Pull flow entries from S

    FEATURE_CREATION(L_E)

  end

  Wait for some time

end

Function `FEATURE_CREATION` (*Flow entries* L_E) **:**

  $F_1 \leftarrow$ Calculate common features for L_E

  forall E *in* L_E **do**

    $F_2 \leftarrow$ Calculate flow-specific features for E

    $F \leftarrow \{F_1, F_2\}$

    ATTACK_DETECTION(F)

  end

Function `ATTACK_DETECTION` (*Feature vector* F) **:**

  $C \leftarrow$ Classify F using M

  if C *is attack* **then**

    $I \leftarrow$ Get source identifiers from F

    MITIGATION(C, I)

  end

Function `MITIGATION` (*Attack type* C*, Identifiers* I) **:**

  if I *is not in* L_B **then**

    $E \leftarrow$ Create flow entry to block or redirect I

    Install E into S

    L_B.add(I)

  end

Algorithm 1: End-to-end Operation

As discussed previously, the RF classifier creates results that are highly explainable to human experts, as opposed to blackbox ML models, whose results are not easily interpretable. For instance, when a specific flow is classified as a DoS attack, it is possible to trace the trees in the forest that voted as a DoS and which feature values caused them to make that decision. This provides the ability for a human network expert to judge the quality of the model, provide recommendations, update the model or take additional actions if necessary.

The Attack Mitigation module is informed by the RF classifier upon attack detection. This module creates a flow rule update depending on the attack type. The created flow rule update is sent to the controller, which installs the flow entries into the corresponding switches. Installed flow entries have higher priority than normal flow entries in the switch. The corresponding action can be dropping the matching packets or redirecting matching flows to a honeypot. Packet blocking and redirection can be based on the source MAC address, source IP or the physical switch port.

CHAPTER 5

SDN DATASETS

In this chapter we provide details of our datasets generated in an SDN-based network simulation. Packet sending rates and packet sizes of normal traffic were taken from a dataset generated in a real testbed. Our datasets contain normal traffic and 5 different attack types, namely DoS, DDoS, port scanning, OS fingerprinting and fuzzing. Normal traffic includes both IoT and regular device traffic. All of our features are SDN specific and can be retrieved using an SDN application in real time. The accuracy of the RF classifier was evaluated with two publicly available SDN datasets we generated and compared with the accuracies of state-of-the-art ML algorithms. Feature selection was used to identify important features for detecting attacks in SDN-based networks. Performance of the model was evaluated for network changes.

We have created 2 SDN datasets [41] and they are available online [42]. Their only difference is the number of IoT devices. In IoT networks, the number of IoT devices may change over time. Our second dataset has more IoT devices and also the number of active IoT devices changed during the traffic recording. The second dataset enables us to evaluate the performance of the models trained with the first dataset. That way we can have an idea about how our model will be affected when the number of IoT devices changes and how often we should update our model.

5.1 Testbed Overview

We used a similar network topology to the Bot-IoT dataset and Figure 5.1 shows the testbed setup of our first dataset. Mininet [43] was used to virtualize our network and an ONOS controller [44] managed our network. An Open vSwitch [45] was used

to connect the controller and simulated devices. During data generation, the server exchanged huge amounts of data with the other two hosts in the system continously. In the first dataset, five hosts in the network simulated IoT services and sent small amounts of data to the server periodically. In the second dataset, initially ten IoT devices were simulated. Two of them were disconnected after a predetermined time. Four attacker hosts targeted the server and IoT devices.

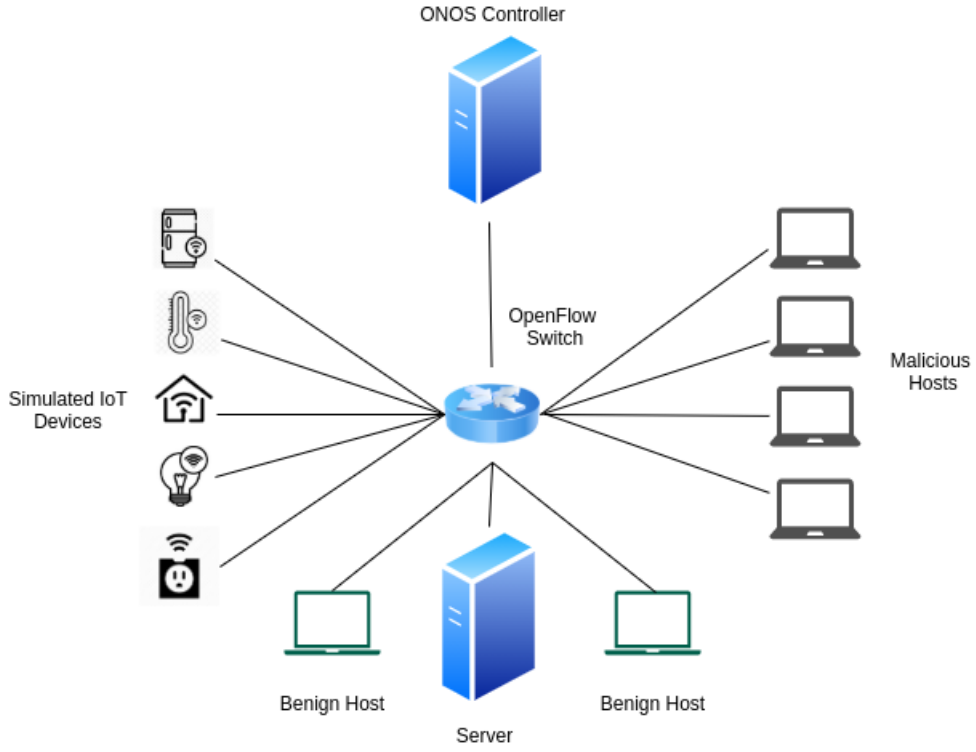

Figure 5.1: Testbed environment for the first dataset.

Our SDN application collected flow entries from the switch at regular intervals and created our features during traffic generation. All of the features in our datasets can be obtained using an SDN application and detection models can be used without affecting the network performance in real time.

5.2 Benign Traffic

We have examined the normal records in the Bot-IoT dataset and created our normal traffic using similar packet sending rates, packet sizes and transport layer protocols.

Bot-IoT dataset has only 9543 records for normal traffic, therefore it was not hard to analyze and replicate the normal traffic. We have generated benign traffic both with and without presence of attack traffic. Without attack scenarios, we have recorded normal traffic 20 times and saved these records into different csv files. Each recording period have lasted 30 to 35 minutes. In the first 15 recording, we have generated both background traffic and IoT traffic. In the last 5 recording, only IoT traffic were generated. During attack scenarios, both background traffic and IoT traffic were generated and recorded. Purpose of saving each scenario into a different csv file is to take equal number of records from each scenario while creating train and test sets and cover different variations of the network traffic without bias.

5.2.1 Background Traffic

For the background traffic, two hosts in the network sent packets to the server constantly. One of the hosts used TCP and the other one used UDP. For TCP traffic, we have used combinations 3 different packet sending rates and 4 different packet sizes. For UDP traffic, we have used combinations 2 different packet sending rates and 2 different packet sizes. Packet sending rates and packet sizes were selected by our packet generation script at the start with the probability proportional to the observation count in the Bot-IoT dataset. For example, the probability of selecting a sending rate of 80 packets per second and a packet size of 1005 bytes for TCP packets was 15.8%.

5.2.2 IoT Traffic

In the Bot-IoT dataset, almost 6000 of the 9543 normal records used UDP and sent only 1 or 2 packets with less than 100 bytes. IoT devices were simulated using Node-red tool and simulated services were weather station, smart fridge, motion activated lights, remotely activated garage door and smart thermostat. All of these services send small amounts of data periodically. We also used these byte counts as packet sizes when we simulated IoT devices in our network. We have used 6 different packet sizes and sent one or two packets each time. In the case of sending two packets, packets

were sent back to back. Our IoT simulation script selected byte and packet counts at the start with a probability proportional to the observation counts in the Bot-IoT dataset. For example, probability of sending one packet with the size of 86 bytes was 16.7% and sending two packets with the size of 88 bytes was 50%. Simulated IoT devices sent packets to the server periodically. ONOS's default flow rule timeout is 10 seconds. If a flow entry does not match any incoming packet for the 10 seconds, the flow entry is deleted. Packet sending periods were also selected randomly from 12 seconds to 15 seconds by our script, which caused every incoming IoT packet to trigger new flow rule installation. Due to the random selection of waiting intervals, the number of flow entries that belong to the IoT devices in the switch was changing dynamically. We used TCP instead of UDP for simulated IoT hosts. The reason is that flow rules are installed after the packet that triggers flow rule installation was forwarded by a packet_out message as explained in the Chapter 2. If we used UDP, we would not be able to collect the statistics of the first packet (duration, byte and packet count). We have used TCP and first packet sent was the SYN message, which is the same for all TCP connections. Therefore, not having SYN packets' statistics was not important.

5.3 Malicious Traffic

Four attacker hosts in our network have launched the cyber-attacks targeting the server or simulated IoT devices. We have performed 5 different attack types: DoS, DDoS, port scanning, OS fingerprinting and fuzzing. Each attack type was performed with different configurations 10 to 12 times depending on the attack type and recording period was 30 to 45 minutes. Each configuration was saved into different csv files. Attack types and tools are explained as follows:

1. **DoS:** Denial-of-service attacks [46] aim to consume target host's and/or network's resources and disrupt the normal functioning of the target host and network. There are two types of this attack: logic and resource attacks. Logic attacks try to find software vulnerabilities and crash the server or decrease its performance. Resource attacks try to drain server's or network's resources such as CPU, memory and bandwidth. The attacker sends lots of forged packets to

the target and tries to make the server unresponsive to the legitimate users. It is not always easy to distinguish the attack packets from the legitimate packets and find the source of the attack. Attackers can easily spoof their IP addresses, MAC addresses and port number and open lots of new connections with the target server. DoS attack with spoofed addresses also causes problems for the SDN controller and switches. Every packet with a spoofed match field (e.g. IP address and source port number) triggers a new flow rule installation. The SDN controller must process all of these requests and install flow rules into the switches. DoS attack with spoofed match fields consume the bandwidth of the channel between the switch and the controller, controller's CPU and memory and also floods the switch's flow tables.

In our network, one attacker host was used to perform DoS attacks targeting the server or an IoT device using the hping3 tool [47]. We have recorded DoS traffic 12 times. Each recording lasted 30 to 45 minutes. In two recordings, the server was targeted with a TCP SYN flooding attack using spoofed IP addresses and source port numbers. In the other two recordings, the server was targeted with a UDP flooding attack using spoofed IP addresses and source port numbers. In one of the recordings, the server was targeted with a TCP SYN attack without spoofing. In one of the recordings, the server was targeted with a UDP flooding attack without spoofing. During the attacks without spoofing, all of the packets coming from the attacker passed over the same flow entry and new flow rule creation was not triggered. In the remaining 6 recordings, the same attack scenarios were performed targeting one of the IoT devices.

Four different packet sending rates were used: 4000, 6000, 8000 and 10000 packets per second. Also four different packet payload sizes were used: 0, 100, 500 and 1000. All of the combinations of these packet sending rates and payload sizes were used.

2. **DDoS:** Distributed denial-of-service attack [48] is a form of denial-of-service attack, where the attacker compromises more than one host and uses the compromised machines to target the same victim. DDoS attack can consume much more resources than DoS because of the additional attacker devices. It is also harder to prevent DDoS attacks and distinguish malicious packets from normal

packets because malicious packets come from different hosts and directions. Compromised hosts might use different attack strategies.

Four attacker hosts were used to perform DDoS attacks targeting the server or an IoT device using hping3 tool. We have recorded DDoS traffic 12 times with the same scenarios as the DoS attacks. Each recording lasted 30 to 45 minutes. We have also used the same packet sending rates and packet sizes as the DoS attacks.

3. **Port scanning:** Port scanning [49] is the process of identifying open ports on the target. The attacker sends packets to the specified range of port numbers with the purpose of identifying services with the known vulnerabilities running using the open ports.

   One attacker host was used to perform port scanning attack targeting the server or an IoT device using nmap tool [50]. Nmap has two modes for port scanning. If you do not specify the port numbers which should be scanned, nmap scans the first 1024 port numbers. These port numbers are called well-known port numbers and reserved for the most commonly used services. You can also specify the port range that you want to be scanned. We have recorded port scanning traffic 10 times. Each recording have lasted 30 to 45 minutes. In two recordings, nmap was used to scan all of the port of the server. In the other two recordings, all of the ports of an IoT device were scanned. In the other three recordings, the first 1024 port numbers of the server were scanned. In the last three recordings, the first 1024 port numbers of an IoT device were scanned.

4. **OS fingerprinting:** OS fingerprinting [51] is the process of identifying the target's operating system type and version. The attacker first discovers the open ports on the target and then sends packets to the victim. The attacker analyzes the content and delay of the response packets and tries to determine OS type and version. OS fingerprinting is usually used for gathering information about the victim and identifying its weaknesses.

   One attacker host was used to perform OS fingerprinting targeting only the server using the nmap tool. We did not target the IoT devices because there were not any open ports on these devices, therefore targeting them would result

in just a port scanning attack. We have recorded OS fingerprinting traffic 10 times. Each recording lasted 30 to 45 minutes.

5. **Fuzzing:** Fuzzing [52] is both a testing method and attack technique, where randomly generated inputs are sent to the target until the target crashes in order to find out potential vulnerabilities, bugs and weaknesses. Fuzzing is a black-box method, so the attacker does not have to know the internal workings of the target system. The attacker examines the input that caused a crash in the target and identifies the vulnerabilities of the system. Then, it can craft special inputs to exploit the weaknesses.

   We have used boofuzz [53] for fuzzing attacks. Boofuzz is installed as a Python library which is used to build fuzzing scripts. We have implemented HTTP and FTP fuzzer scripts. HTTP and FTP server code were running on our server. One of the attacker hosts were used to perform HTTP and FTP fuzzing attacks against the server. We have recorded fuzzing attacks 10 times. In the first five recordings, FTP fuzzing was used. In the last five recordings, HTTP fuzzing was used.

5.4 Flow Collection and Feature Generation

Most of the existing datasets have been created by processing pcap files after traffic had been generated. Our purpose was to create a dataset specific to SDN and use this dataset to prevent attacks using the SDN controller in real time. Therefore, we developed an SDN application for retrieving flow entries from the switch every 1 second and creating our features for each flow. Our application labeled each flow using the ports that hosts are connected to the switch and wrote the features into csv files. Every recording session of every attack type saved into different csv files.

We tried to create all of the features in the Bot-IoT dataset. However, in the Bot-IoT dataset, authors used Argus tool to extract features. Our SDN application only used the flow entry fields pulled from the switch, hence all of the features in our dataset are SDN specific and can be calculated using an SDN application. We were not able to create features in the Bot-IoT that are not specific to SDN environments (e.g. argus

sequence number).

Our datasets contain 33 features. Our feature names and describtions are explained in the Table 5.1. We have features that are specific to each flow (e.g. duration, source-to-destination packet and byte count). We also have features that are specific to each flow but containing statistics about the reverse of the flow (e.g. destination-to-source packet per second and packet and byte count). In addition, we have common features for every flow in the switch that were calculated using all of the flow entries (e.g. standard deviation of flow durations, number of inbound connections per source and destination IP and total number of packets per source and destination IP). Two of the features are labels: attack and category. Attack shows whether flow entry belongs to an attack scenario or not. Attack label can be used for binary classification. On the other hand, category shows which traffic category flow entries belong to: normal, DoS, DDoS, port scanning, OS fingerprinting and fuzzing. Category label can be used for multiclass classification.

Normal traffic were recorded during all of the attack scenarios and without attack scenarios. DoS and DDoS attacks with spoofed IP addresses created huge number of duplicate flow entries. Therefore, we limited saved record count to 100 every time flow entries were pulled from the switch for DoS and DDoS attacks. Port scanning and OS fingerprinting attacks also created huge number of flow entries but we did not limit them because created packets were not duplicates. Our first SDN dataset contains 27.9 million entries. Table 5.2 shows the entry counts and percentages for every category.

Our second SDN dataset were created using the same scenarios. Only difference was the higher number of IoT devices as explained before. Our second SDN dataset contains 30.2 million entries. Table 5.3 shows the entry counts and percentages for every category.

Feature retrieval time is very important. There is no point detecting attacks after they are over or caused severe damage. Features should be retrieved quickly for efficient attack detection and prevention. Also feature retrieval process should not consume a lot of controller resources, otherwise network performance would be negatively affected. Figure 5.2 shows the flow entry collection and feature creation time up to

Table 5.1: Features of SDN Datasets

Feature	Description
srcMac	Source MAC address
dstMac	Destination MAC address
srcIP	Source IP address
dstIP	Destination IP address
srcPort	Source port number
dstPort	Destination port number
last_seen	Record last time
Protocol	Textural representation of transaction protocol
proto_number	Numerical representation of feature proto
Dur	Record total duration
Mean	Average duration of aggregated records
Stddev	Standard deviation of aggregated records
Min	Minimum duration of aggregated records
Max	Maximum duration of aggregated records
Pkts	Total count of packets in transaction
Bytes	Total number of bytes in transaction
Spkts	Source-to-destination packet count
Dpkts	Destination-to-source packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count
Srate	Source-to-destination packets per second
Drate	Destination-to-source packets per second
Sum	Total duration of aggregated records
TnBPSrcIP	Total number of bytes per source IP
TnBPDstIP	Total number of bytes per destination IP
TnP_PSrcIP	Total number of packets per source IP
TnP_PDstIP	Total number of packets per destination IP
TnP_PerProto	Total number of packets per protocol
TnP_Per_Dport	Total number of packets per destination port
N_IN_Conn_P_SrcIP	Number of inbound connections per source IP
N_IN_Conn_P_DstIP	Number of inbound connections per destination IP
Attack	Attack or not
Category	Traffic category

Table 5.2: Distribution of records in the first SDN dataset

Category	Size	%
Normal	1.67M	5.99
DoS	793K	2.84
DDoS	192K	0.67
Port Scanning	20.68M	74.08
OS and Service Detection	3.39 M	12.15
Fuzzing	1.18M	4.24

Table 5.3: Distribution of records in the second SDN dataset

Category	Size	%
Normal	2.67M	8.84
DoS	495K	1.67
DDoS	182K	0.60
Port Scanning	22.44M	74.23
OS and Service Detection	3.39 M	11.20
Fuzzing	1.05M	3.48

1000 flow entries in the switch, which corresponds to the normal traffic. When switch had 1000 flow entries, flow collection and feature creation time for all of the flows was 22.8 milliseconds, which is fairly low.

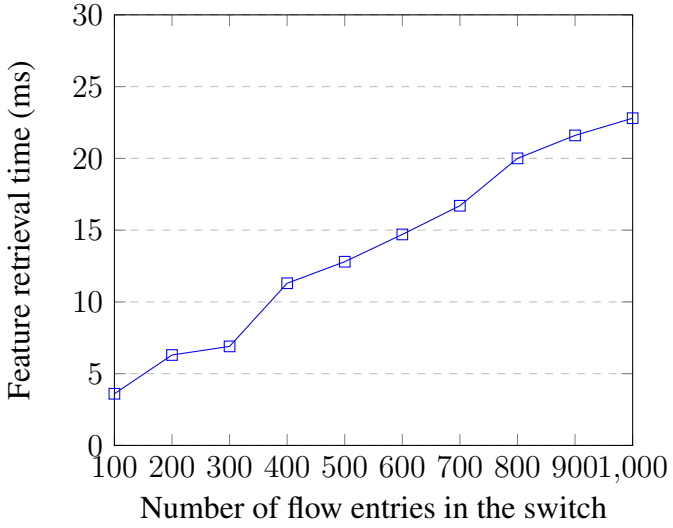

Figure 5.2: Feature retrieval time up to 1000 flow entries.

Figure 5.3 shows the flow entry collection and feature creation time up to 20000 flow entries in the switch, which corresponds to the attack traffic. Even though there were 20000 flow entries in the switch, our SDN application collected flow entries and created features for all the flows in 411.3 milliseconds, which does not cause a lot of overhead for our controller. Feature retrieval time increases linearly with the number of flow entries in the switch and our algorithm is O(n).

5.5 Model Evaluation

In this section, we describe how we obtained train and test datasets from created SDN datasets and share the results of different machine learning algorithms.

5.5.1 Pre-processing

Our datasets contain millions of records and record counts belonging to every category is not the same as shown before in Table 5.2 and Table 5.3. Processing millions

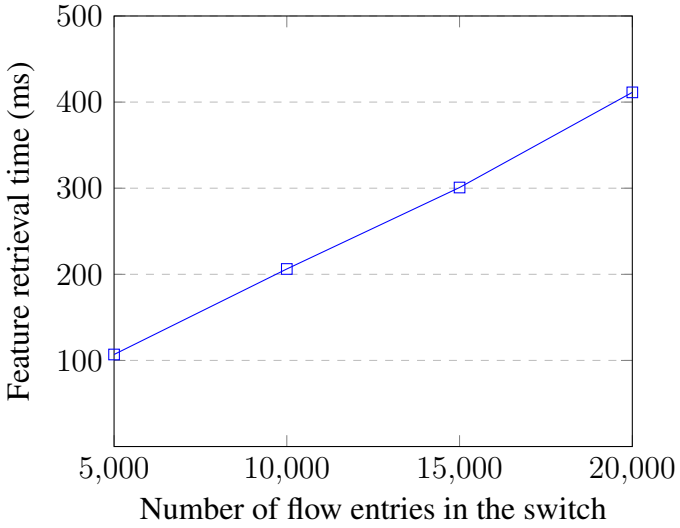

Figure 5.3: Feature retrieval time up to 20000 flow entries.

of data records is not feasible and may lead to overfitting. Also imbalanced datasets might cause biased models. Around 74% of the records belong to the port scanning attack. Therefore, we wanted to take an equal number of records from every category for model training. We also wanted to take an equal number of records from each recording of a category. The reason is that, depending on the configuration and target, record counts differed a lot. For example, one of the DoS attacks without spoofing had the lowest record count of 3251, while DoS attacks with spoofing had up to 137000 records. We recorded DoS traffic 12 times, so the maximum number of records we could get was 39012. Therefore, we took 35000 records from every attack category taken equally from every scenario of that attack type, resulting in a total of 175000 attack records.

For multi-class classification, we also took 35000 normal records. Normal records were taken equally from the DoS, DDoS, port scanning, OS fingerprinting, fuzzing and normal traffic without attack files, 5834 each. The constructed dataset had 35000 records from every category, with a total of 210000 records. We split this dataset into training and test sets. The training set has 25000 records from every category, with a total of 150000 records. The test set had 10000 records from every category, with a total of 60000 records.

For binary classification, we took 175000 normal records equal to the total number

of attack records as explained above. The constructed dataset has 350000 records in total. We split this dataset into training and test sets. The training set had 250000 records and test set had 100000 records.

The same procedure was followed for both of the datasets and training and test datasets were created for both.

5.5.2 Performance Metrics

For the classification, most commonly used metrics in the literature are accuracy, presicion, recall and F1 score. Figure 5.4 shows the confusion matrix (error matrix) and required definitions for metric calculations.

		Predicted Class: Positive	Predicted Class: Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 5.4: Confusion matrix.

- **Accuracy:** Ratio of correctly classified positive and negative records to the total number of records.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

- **Precision:** Ratio of correctly classified positive records to the total records classified as positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall:** Ratio of correctly classified positive records to the total positive records.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1 Score:** Balanced metric between presicion and recall.

$$F_1 = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{5.4}$$

Performance metrics should be chosen based on our goals and system requirements. Precision is a good metric when we want to lower the amount of false positives. On the other hand, recall performs well when we want to lower the amount of false negatives. F1 score gives us a balance between precision and recall. Therefore, we chose F1 score as our evaluation metric in our experiments.

5.5.3 Multi-class Classification

The constructed training and test sets were used to evaluate the performances of different machine learning algorithms. We have used all of the features except host identifiers: srcMac, dstMac, srcIP, dstIP, srcPort, dstPort, last_seen and proto_number. Different machine learning algorithms were trained and tested using the first SDN dataset's training and test sets. The results of multi-class classification of different algorithms are shown in Figure 5.5: naive bayes (NB), logistic regression (LR), k-nearest neightbour (K-NN), support vector machines (SVM), kernel support vector machines (K-SVM), random forest (RF) and XGBoost (XGB). RF and XGB performed better than the other algorithms.

Our goal of creating two datasets was to test on the second dataset using the models trained with the first dataset and see how the system would be affected from network changes. When we evaluated the models on the second dataset's test set using the trained random forest and XGBoost models, both of the models achieved around 80% F1 score, which is much lower than the performance on the first dataset. We were using all of the training set which contains 25000 records for each category. When we decreased this number to 10000 records for each category, performances of both models increased a bit. We applied feature selection based on the feature importance

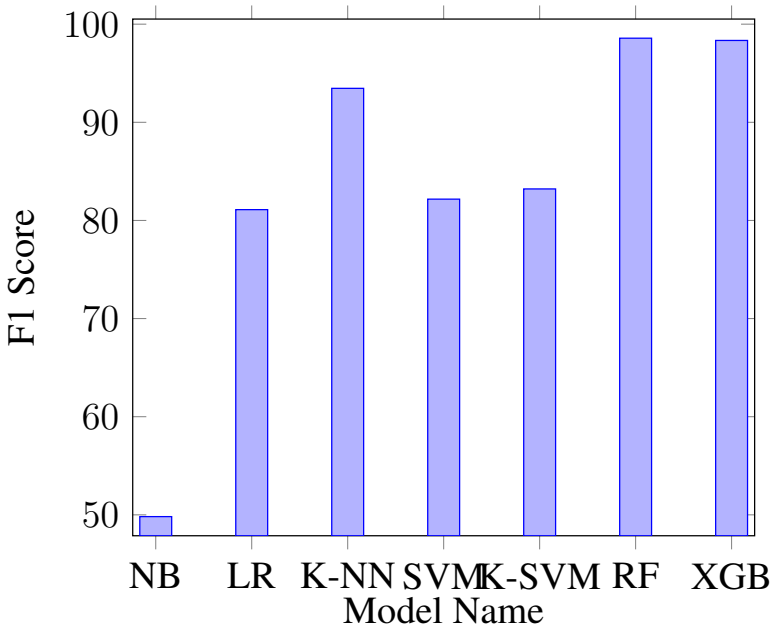

Figure 5.5: Model comparison.

attribute of random forest and XGBoost algorithms. The feature importance attribute returns impurity-based feature importance of each feature on the training set. We used features that had higher feature importance than the average of the feature importance values. The performance of the XGBoost model on the second dataset decreased a lot and performance of the random forest model increased. Therefore, we focused on the random forest model. We also added one feature whose importance was close to the average and ended up with 10 features. The selected features and their descriptions are shown in Table 5.4.

The overall F1 score of the RF model, trained with the selected features, on the first dataset was 97.86%. Performance was still close to the model trained with all of the training data and 24 features even though we reduced both training data and number of features by more than half. Using less features also allows our SDN application to retrieve features much more quickly. Performance metrics for all classes in the first dataset are shown in Table 5.5.

Normal traffic had the lowest F1 score, which is not desirable, as we do not want to classify normal packets as malicious packets and block legitimate traffic. Five over six of the normal records in our test set belonged to the normal traffic during attack

Table 5.4: Initially selected features.

Feature	Description
Dur	Record total duration
Mean	Average duration of aggregated records
Spkts	Source-to-destination packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count
Sum	Total duration of aggregated records
TnP_PSrcIP	Total number of packets per source IP
TnP_Per_Dport	Total number of packets per destination port
N_IN_Conn_P_SrcIP	Number of inbound connections per source IP
N_IN_Conn_P_DstIP	Number of inbound connections per destination IP

Table 5.5: Performance of RF using initially selected 10 features.

Class	F1 Score	Precision	Recall
Normal	94.69	96.71	92.75
DoS	98.37	98.08	98.66
DDoS	98.03	97.21	98.87
Port scanning	98.86	98.54	99.19
OS and service detection	98.36	98.35	98.37
Fuzzing	98.86	98.32	99.41

scenarios. Distinguishing normal traffic from attack traffic during an attack is not an easy task. Therefore, we must be sure before taking action upon detecting an attack. In the absence of any attack traffic, our model's accuracy of detecting normal traffic was 99.67%, which is quite high.

The overall F1 score on the second dataset was 84.48% using the initially selected 10 features. Two features were replaced and the overall performance on the second dataset increased to 91.01% while using the features listed in Table 5.6 and hyper-parameters listed in Table 5.7. Our model's performance for the normal traffic on the second dataset was similar to the performance on the first dataset. However, the overall performance was lower than the first dataset, because our model classified some of the DoS attacks as DDoS attacks on the second dataset as expected, due to the increased number of IoT devices in the second dataset. As the mitigation action taken is the same, the network performance is not affected. Changing the features did not affect the model's performance much on the first dataset and resulted in 97.82% overall F1 score.

Table 5.6: 10 best features for multi-class classification.

Feature	Description
Dur	Record total duration
Mean	Average duration of aggregated records
Stddev	Standard deviation of aggregated records
Spkts	Source-to-destination packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count
Sum	Total duration of aggregated records
TnP_PSrcIP	Total number of packets per source IP
TnP_PDstIP	Total number of packets per destination IP
TnP_Per_Dport	Total number of packets per destination port

F1 scores for every class in the first dataset are shown in Table 5.8. Results are similar to the initially selected features. On the other hand, using the features in Table 5.6

Table 5.7: Hyperparameters of Random Forest model for multi-class classification.

Hyperparameter	Value
n_estimators	100
criterion	entropy
max_depth	40
max_features	auto
min_samples_leaf	1
min_samples_split	6
bootstrap	True
random_state	0

performed well both on the first and second datasets.

Table 5.8: Performance of RF using 10 best features.

Class	F1 Score	Precision	Recall
Normal	94.80	96.75	92.92
DoS	98.44	97.77	99.12
DDoS	97.93	97.29	98.58
Port scanning	98.89	98.52	99.27
OS and service detection	98.08	98.40	97.76
Fuzzing	98.76	98.20	99.32

5.5.4 Binary Classification

Same training and test datasets used in multi-class classification were used for binary classification. All of the 25000 normal records in the training set were taken and combined with 5000 records from each of the 5 attack categories to make it a balanced training set for binary classification. Random forest classifier were used and the same

feature selection and hyperparameter selection processes were performed. Our binary classification model used 12 features shown in table 5.9. Among these 12 features, 7 of them are common with multi-class model. Binary classification model used all of the flow specific features (Dur, Spkts, Dpkts, Sbytes, Dbytes, Srate, Drate), unlike multi-class classification model which used only 4 of them (Dur, Spkts, Sbytes, Dbytes). Both of the models used Dur, Mean and Stddev features. Remaining features used by the models are different.

Table 5.9: 12 best features for binary classification.

Feature	Description
Dur	Record total duration
Mean	Average duration of aggregated records
Stddev	Standard deviation of aggregated records
Spkts	Source-to-destination packet count
Dpkts	Destination-to-source packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count
Srate	Source-to-destination packets per second
Drate	Destination-to-source packets per second
TnP_Per_Dport	Total number of packets per destination port
N_IN_Conn_P_DstIP	Number of inbound connections per source IP
N_IN_Conn_P_SrcIP	Number of inbound connections per destination IP

Grid search was performed to select best hyperparameters. Table 5.10 shows the best hyperparameters for our binary classification model. Best hyperparameters of binary classification model are similar to the best hyper parameters of multi-class classification model. Only values of max_features and min_samples_leaf hyperparameters are different.

Binary classification model achieved overall 97.64% F1 score which is 0.2% lower than the performance of multi-class classification model. However, binary classification model achieved 100% accuracy for normal traffic in the absence of attack traffic.

Table 5.10: Hyperparameters of Random Forest model for binary classification.

Hyperparameter	Value
n_estimators	100
criterion	entropy
max_depth	40
max_features	log2
min_samples_leaf	6
min_samples_split	6
bootstrap	True
random_state	0

Also, binary classification model achieved 97.01% overall F1 score when we tested it on our second dataset. Multi-class classification model's performance on the second dataset was 91.01%. We were expecting this performance difference on the second dataset between models because multi-class classification model misclassified some of the DoS records as DDoS. In binary classification, all of the malicious traffic were classified as attack.

5.5.5 Selected Features

Selected features for multi-class and binary classification are below. Some of the features are common for all of the flow entries in the switch. Common features are used to detect the presence of the attacks and the remaining features are used to distinguish benign flows from the malicious ones.

- **Dur, Mean, Stddev and Sum:** Statistics about the duration of the flow entries in the switch are used to detect attacks. During the attacks, lots of new flow entries are installed into the switches. Newly installed flow entries decrease the mean and stddev of the duration of flow rules. Therefore, these features are important to detect the presence of the attack in the network. On the other

hand, duration helps to identify malicious and benign flows in the presence and absence of the attacks.

- **Spkts, Srate and Sbytes** Source-to-destination packet and byte counts are low for the most of the attack flows. Most of the time attackers use spoofing to make it harder to detect the source of the attack. Every incoming attack with spoofed addresses trigger a new flow entry installation and only 1 or 2 packets passes over these flow entries depending on the attack type. Also bytes of these packets are usually low in order to send more packets to the target. In DoS and DDoS attacks without spoofing, all of the attack packets passes over the same flow entry and source-to-destination packet count, packet rate and byte counts increase dramatically.

- **Dpkts, Drate and Dbytes** Reverse flow statistics are also important. Reverse flow values for the attack flows is usually lower than the benign flows. Most of the time controller does not know how to route the packet to the spoofed address of the attacker since the spoofed address does not exist in the network.

- **TnP_Per_Dport** Total number of packets per destination port is another important feature for detecting the attacks. If the target of DDoS and DDoS attack is a web server, all of the packets are sent to the same port. Therefore, number of packets per destination port increases. On the other hand, during the port scanning attack, most of the packets are sent to the different ports of the target and number of packets per destination port decreases.

- **N_IN_Conn_P_DstIP** Number of inbound connections per destination IP increases a lot if the attackers use spoofed source addresses. All of the incoming packets from the attackers trigger a new flow entry installation with the same destination IP.

- **N_IN_Conn_P_SrcIP** Number of inbound connections per source IP is also useful for detecting attacks with spoofed source addresses. Spoofed IP addresses probably do not exist in the network, therefore most of the time controller can not install reverse flows for the attack packets. Number of inbound connections per source IP decreases during the attacks with spoofed source addresses.

- **TnP_PSrcIP** Similar to the N_IN_Conn_P_SrcIP, total number of packets per source IP is another important feature for attack detection. Its value depends on the spoofing. If attackers do not use spoofing, value of this feature increases a lot. With spoofing, the value is lower because of the increased flow entry count.
- **TnP_PDstIP** Total number of packets per destination IP is a useful feature that can detect attacks with and without spoofing. Attackers send lots of packets to the target IP address and number of packets per destination IP increases rapidly.

CHAPTER 6

EXPERIMENTAL EVALUATION

In this chapter we provide an experimental evaluation of the proposed security approach using an SDN-managed network simulation environment. We performed experiments to evaluate the end-to-end intrusion detection and mitigation system in terms of its effect on the network performance during DoS attacks of different types. The experiments were conducted on a machine with Intel Core i7-8750H @ 2.20GHz processor and 16 GB RAM.

6.1 Experiment Setup

For the deployment of the proposed intrusion detection and mitigation system, same network topology of first SDN dataset, shown in Figure 5.1 before, was used. Mininet was used to virtualize the network. In addition, maximum bandwidth of each link between hosts and the switch was limited to 100 Mb per second. An ONOS controller managed the network. Simulated IoT devices, benign hosts and server transmitted data as explained in the SDN Datasets section.

Some attack types also affect the performance of network as well as the target. DoS and DDoS attacks decrease available bandwidth and consume resources of the controller and switches. Other attack types in our dataset do not have a significant effect on the network. Their purpose is to find vulnerabilities of the target and crash it if possible. Therefore, we focused on DoS and DDoS attack in our network performance experiments. One malicious host was used to perform DoS attacks and effects of the attacks on the network were measured.

The Onos controller was configured to pull flow entries from the switch every second. Our SDN application retrieved flow entries from the controller and generated the 10 best features required by our multi-class random forest classifier for each flow entry. Algorithm 2 and Algorithm 3 show the steps of feature creation. Our application calculated the common features first with an initial pass over the flow entries. Common features were Mean, Stddev, Sum, TnP_PSrcIP, TnP_PDstIP and TnP_Per_Dport. Their descriptions are in Table 5.6. Hash sets were used to store unique source IPs, destination IPs and destination port numbers. A list was used to store duration of flow entries. While looping over the flow entries, packet count of the flow entries were added to the total packet count. Duration of the flow entries were added to the duration list. Source IPs, destination IPs and destination port numbers were added to the corresponding hash sets. Byte count of the flows were added to a hash map. Keys of this map was made of source IP, source port, destination IP and destination port for TCP and UDP packets. For ICMP packets, keys were made of source IP and destination IP since they do not have port numbers. This map was later used for retrieving reverse flow statistics. After looping over all of the flow entries, common features were calculated using the total packet count, hash sets and duration list. Flow specific features, i.e. Dur, Spkts, Sbytes and Dbytes, were calculated within the second pass over the flow entries. Duration, packet count and byte count of the flow entries were extracted. Hash map that had been created in the common feature creation was used to retrieve destination to source byte count. After creating the feature vector for a flow entry, it is sent to classification without waiting creation of other feature vectors. The SDN application waited for 1 second after creating features for all of the flow entries in the switch and then continued to create features by retrieving new flow entries from the controller.

In the SDN Datasets section, we reported performances of classifiers implemented using Python. However, SDN applications are written in Java. Therefore, we implemented our Random Forest model in Java using the same features. Overall F1 score of the model implemented in Java was 97.80. This score is quite close to the F1 score of model implemented in Python, which was 97.82.

Our model classified every flow entry. In order to prevent false positives, we used 3 as a threshold. When our application detected the third attack flow coming from a

Input: Flow entries

$L_E \leftarrow$ Flow entries

$C \leftarrow$ CALCULATE_COMMON(L_E)

forall *E in L_E* **do**

- $F.Sbytes \leftarrow E$.getByteCount()
- $F.Spkts \leftarrow E$.getPacketCount()
- $F.Dur \leftarrow E$.getDuration()
- $F.Mean \leftarrow C.mean$
- $F.Stddev \leftarrow C.stddev$
- $F.Sum \leftarrow C.sum$
- $F.TnP\_PSrcIP \leftarrow C.TnP\_PSrcIP$
- $F.TnP\_PDstIP \leftarrow C.TnP\_PDstIP$
- $F.TnP\_Per\_Dport \leftarrow C.TnP\_Per\_Dport$
- $srcIP \leftarrow E$.getSourceIp()
- $dstIP \leftarrow E$.getDestinationIp()
- $P \leftarrow E$.getSwitchPort()
- $proto\_number \leftarrow E$.getInternetProtocolNumber()
- **if** *$proto\_number$ is TCP or UDP* **then**
  - $srcPort \leftarrow E$.getSourcePort()
  - $dstPort \leftarrow E$.getDestinationPort()
  - $key \leftarrow$ dstIP + dstPort + srcIP + srcPort
  - $F.Dbytes \leftarrow C.hashMap$.get(key)
- **end**
- **else if** *$proto\_number$ is ICMP* **then**
  - $key \leftarrow$ dstIP + srcIP
  - $F.Dbytes \leftarrow C.hashMap$.get(key)
- **end**

end

Algorithm 2: Feature Creation

Function CALCULATE_COMMON (*Flow entries* L_E) **:**

$C \leftarrow$ Common statistics

$srcIpSet \leftarrow$ Create source IP HashSet

$dstIpSet \leftarrow$ Create destination IP HashSet

$portSet \leftarrow$ Create destination port HashSet

$L_D \leftarrow$ Create duration List

forall E *in* L_E **do**

$pkts \leftarrow E$.getPacketCount()

$totalPacketCnt \leftarrow totalPacketCnt$ + $pkts$

$bytes \leftarrow E$.getByteCount()

L_D.add(E.getDuration())

$srcIP \leftarrow E$.getSourceIp()

$srcIpSet$.add($srcIP$)

$dstIP \leftarrow E$.getDestinationIp()

$dstIpSet$.add($dstIP$)

$proto\_number \leftarrow E$.getInternetProtocolNumber()

if $proto\_number$ *is TCP or UDP* **then**

$srcPort \leftarrow E$.getSourcePort()

$dstPort \leftarrow E$.getDestinationPort()

$key \leftarrow$ srcIP + srcPort + dstIP + dstPort

$C.hashMap$.put(key,bytes)

end

else if $proto\_number$ *is ICMP* **then**

$key \leftarrow$ srcIP + dstIP

$C.hashMap$.put(key,bytes)

end

end

$C.tnP\_PSrcIp \leftarrow totalPacketCnt/srcIpSet.size()$

$C.tnP\_PDstIp \leftarrow totalPacketCnt/dstIpSet.size()$

$C.tnP\_Per\_DPort \leftarrow totalPacketCnt/portSet.size()$

$C.mean$, $C.stddev$, $C.sum \leftarrow$ mean, standard deviation and sum of L_D

return C

Algorithm 3: Calculation of common statistics and features

switch port, the mitigation process started. If an attack was detected, the attacker was blocked based on the physical port through which it was connected to the switch. Our application installed a flow rule into the switch in order to drop the packets coming from the switch port that attack was connected to. The installed flow rule had a priority of 1000, which is higher than the default flow rule priority (10). Figure 6.1 shows a flow rule installed by our application to drop the packets coming from port 1, and figure 6.2 shows a flow rule for a packet classified as normal. "Selector" shows the packet match fields and their values. "Immediate" field of the "treatment" shows the action upon matched packets. If "OUTPUT" is specified, packets are forwarded to the specified switch port. "NOACTION" means dropping the packet.

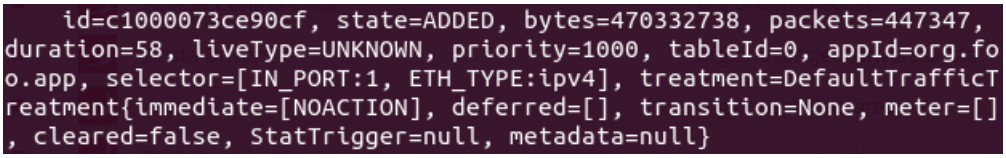

Figure 6.1: Installed flow rule for dropping packets.

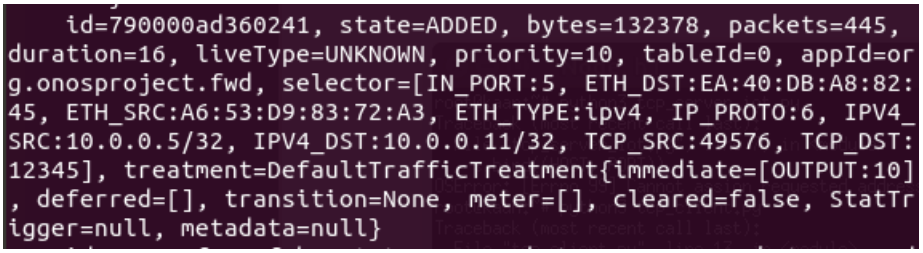

Figure 6.2: Installed flow rule for normal packets.

6.2 Network Performance Results

In the following subsections, performance measurements of our intrusion detection and mitigation system are reported.

6.2.1 Time Measurements

We measured the feature retrieval time and also feature retrieval and classification time using our SDN application. Counter was started before our application pulled

flow entries from the switch and stopped when feature calculation and classification was over for all of flow entries in the switch.

Figure 6.3 and Figure 6.4 show feature retrieval times of our 10 best features used by the multi-class Random Forest model. Figure 6.3 corresponds to the network without presence of attacks. Figure 6.4 corresponds to the network under a DoS attack.

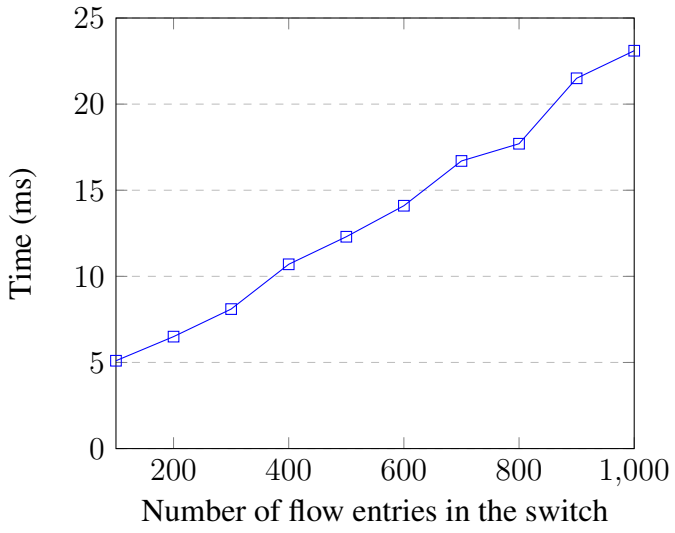

Figure 6.3: Feature retrieval time of 10 best features up to 1000 flow entries.

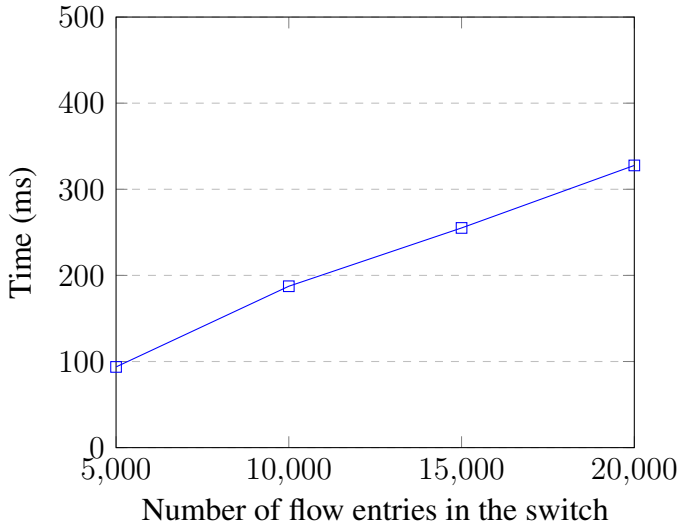

Figure 6.4: Feature retrieval time of 10 best features up to 20000 flow entries.

Feature retrieval time of all features for 20000 flow entries was 411 milliseconds, whereas it was 327 milliseconds for retrieving the 10 best features. When our ap-

plication calculates the common features, it also creates a hash map that uses source IP, source port, destination IP and destination port as the key and byte count, packet count and packet rate as values. This map is later used for getting reverse flow statistics, i.e. destination-to-source packet count, byte count and packet rate. Converting source and destination IP to string for key of the map takes a long time. Our model uses destination-to-source byte count (Dbytes) as a feature as shown in the Table 5.6. This is the reason why improvement on the feature retrieval time was not much.

Figure 6.5 shows feature retrieval times of the 10 best features and classification time for up to 1000 flow entries in the switch. It is fairly low and does not affect the performance of the network. Figure 6.6 shows feature retrieval times of the 10 best features and classification time for up to 20000 flow entries in the switch, which corresponds to the DoS attack with spoofed addresses. Feature retrieval and classification take around 900 milliseconds for 20000 flow entries. However, the SDN application does not wait to finish classifying every flow entry in the switch before taking action. Attackers are blocked immediately when they reach the detection threshold. Therefore, most of the time, attacks are mitigated before a huge number of attack flows are installed into the switch.

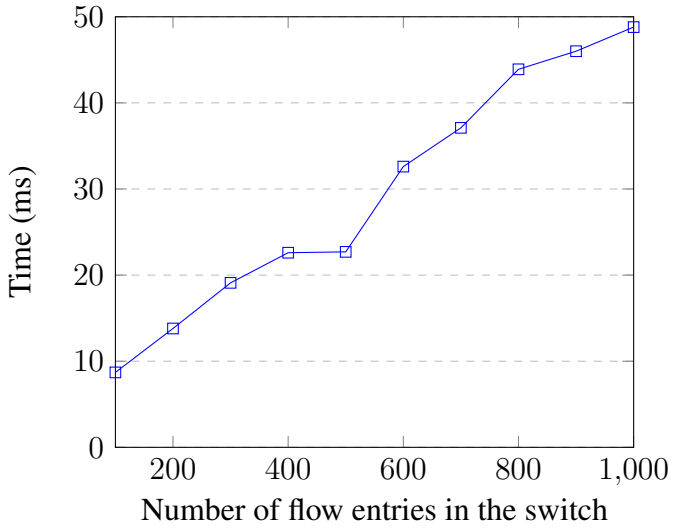

Figure 6.5: Feature retrieval time of 10 best features and classification time up to 1000 flow entries.

Our application calculates feature vectors and classifies them in 9 milliseconds when

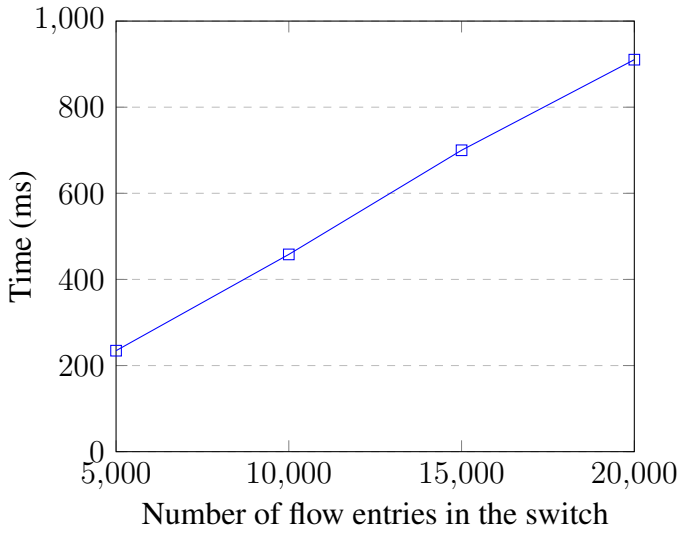

Figure 6.6: Feature retrieval time of 10 best features and classification up to 20000 flow entries.

switch has 100 flow entries. This procedure takes 49 milliseconds when switch has 1000 flow entries. We believe these times are fairly low and does not affect normal operation of the controller and the network. Under DoS attack, feature vector calculation and classification takes less than a second for all of the 20000 flow entries. Flow entries are classified one by one and mitigation is performed immediately upon attack detection. Therefore, attacks are swiftly mitigated before they can do serious damage to the target and the network.

6.2.2 Bandwidth Measurements

The maximum available bandwidth of all of the links between the switch and the hosts in our network were set to 100 Mb per second. The iPerf3 tool [54] was used to measure the available bandwidth between one of the IoT devices and the server with and without presence of DoS attacks. One malicious host was used to perform a DoS attack targeting the server. The packet sending rate was 1000 packets per second and the payload of the packets was 1000 bytes. Attacks started after 5 seconds.

Figure 6.7 shows the available bandwidth under TCP SYN flood attack without spoofing. All of the packets coming from the attacker passed over the same flow entry in

case of no spoofing. Therefore, it took 3 detection processes to exceed the threshold. Available bandwidth between one of the IoT devices and the server was around 95 Mb per second during the normal operation of the network. Without protection, bandwidth decreased to 37 Mb per second. When our protection was active, the attacker was blocked based on the physical port after exceeding the threshold. Bandwidth returned back to normal after a couple of seconds.

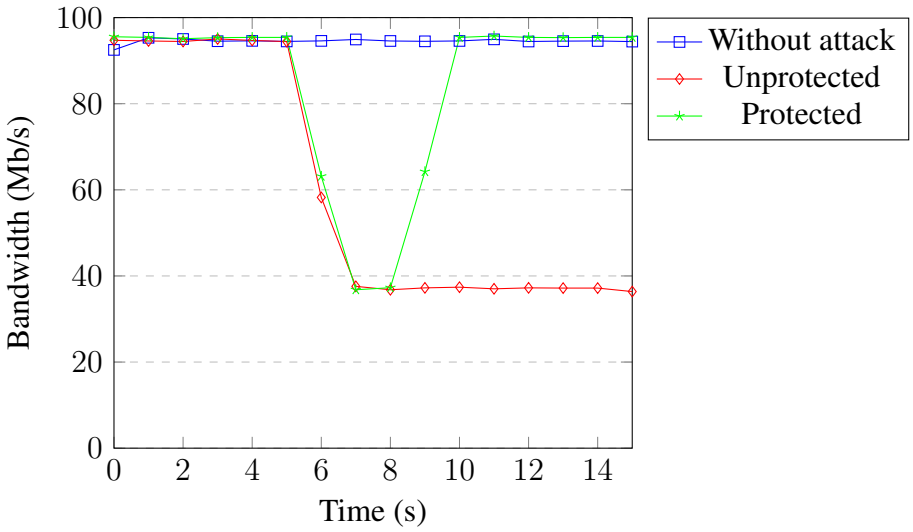

Figure 6.7: Available bandwidth under SYN flood without spoofing.

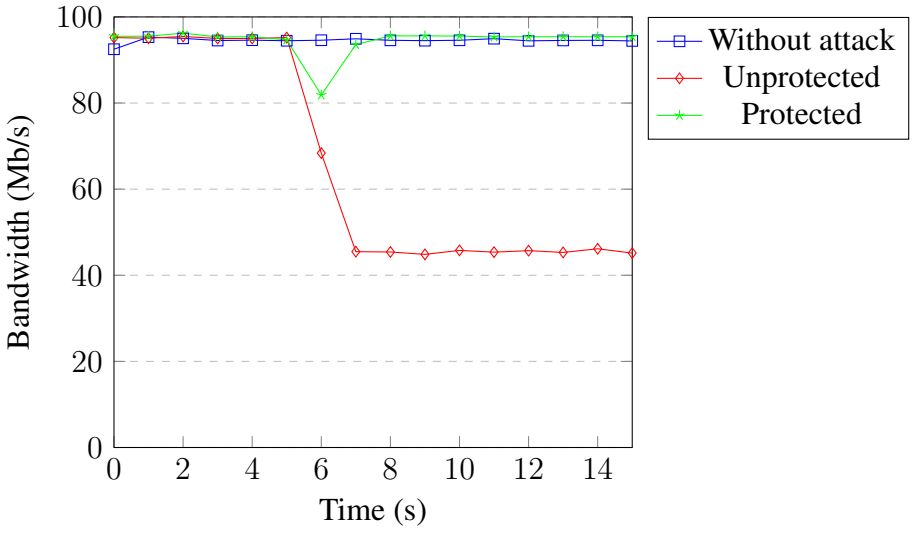

Figure 6.8: Available bandwidth under SYN flood with spoofing.

Figure 6.8 shows the available bandwidth under TCP SYN flood attack with spoof-

ing. Every packet coming from the attacker that missed the flow rules in the switch caused a new flow rule installation. This process slowed the forwarding of malicious packets. Available bandwidth under attack decreased to 45 Mb per second. When our protection was active, bandwidth decreased to 82 Mb per second only for a second and then returned back to normal. The threshold was exceeded in the first detection process and the attacker was blocked immediately.

Figure 6.9 and Figure 6.10 shows the available bandwidth under UDP flood attack with and without spoofing. Results are similar to the TCP SYN flood attack.

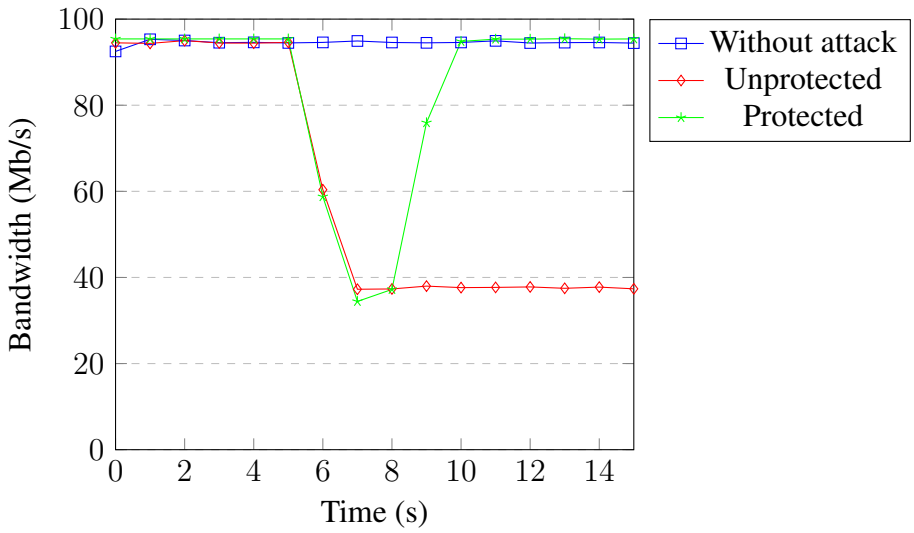

Figure 6.9: Available bandwidth under UDP flood without spoofing.

Overall, available bandwidth returned back to normal within 1 to 3 seconds depending on the attack properties when our protection was active. Our protection quickly prevents attackers to cause damage to the target and the networks. For the DoS attacks with spoofed addresses, attackers are detected within a second and network recovers immediately. For the DoS attacks without spoofed addresses, our application waits until attackers reach the threshold (around 3 seconds) and then blocks the attackers. Network recovers in 1-2 seconds after detection.

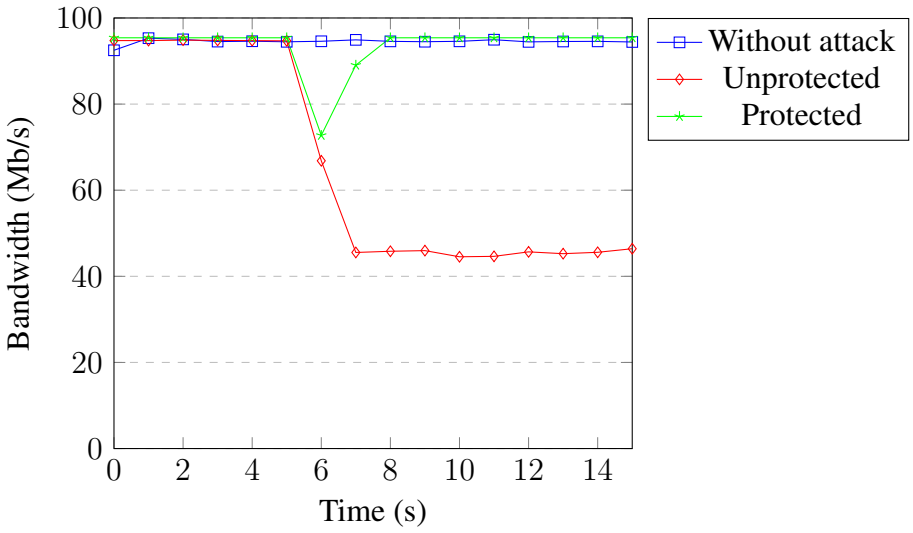

Figure 6.10: Available bandwidth under UDP flood with spoofing.

6.2.3 CPU Measurements

DoS and DDoS attacks with spoofed addresses waste resources of both the controller and the switches. Spoofed match fields of the attack packets cause "table miss" events for each packet. Switches buffer these packets and send a packet_in message to the controller for every attack packet. The controller processes these packets and decides the route. The controller sends a packet_out message to the switch, which contains the determined action for the packet. The controller also installs flow rules for every attack packet.

The ONOS controller and switch were running on the same machine in these experiments. Therefore, we used the Linux top command to measure CPU usages of their processes. Our machine had 6 cores and 2 threads per core, which makes maximum CPU utilization 1200%. One malicious host was used to perform the DoS attack with spoofed IP addresses. Packet sending rate was 1000 packets per second and payload of the packets was 0 byte. Attacks started after 5 seconds. Under normal conditions, most of the time CPU utilization of the hosts were 0%. Rarely, CPU utilization of the 2 benign hosts that sent data to the server were 5.9-6.1%. During DoS attacks, CPU utilization of the attacker was around 30%.

Figure 6.11 shows the CPU usage of the controller under TCP SYN flood attack.

CPU usage was around 2% for our normal network traffic. During the attack without protection, CPU utilization reached 500% within two seconds and stayed there for 7-8 seconds. Then, it dropped to 400%. When our protection was active, CPU utilization reached 180% for a second and then dropped to around 35% for the next 15 seconds. Then, CPU utilization returned to normal. Even though the attacker was blocked in the first detection process, attack flows were installed into the switch until the controller installed the block rule. Our classification model kept classifying them in the following detection processes until these flow entries timed out. This is the reason why CPU utilization remained around 35% for a short time.

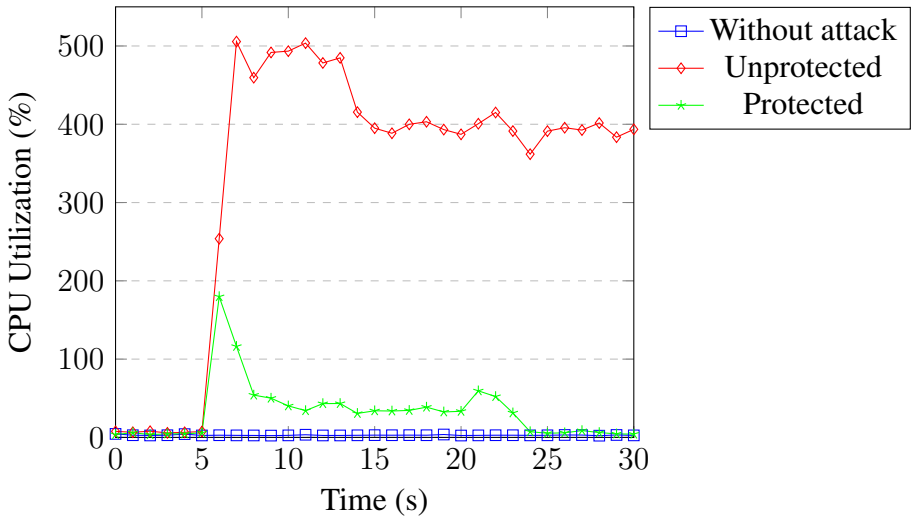

Figure 6.11: Controller CPU under SYN flood.

Figure 6.12 shows the CPU usage of the switch under TCP SYN flood attack. CPU usage was around 1% for our normal network traffic. During the attack without protection, CPU utilization of the switch process reached around 370%. When our protection was active, CPU utilization increased to 135% for a second and then returned back to normal within a a couple of seconds. The controller installed the flow block rule in the first detection process and all of the packets coming from the attacker were dropped by matching the installed flow rule.

Figure 6.13 and Figure 6.14 show CPU utilization of the controller and the switch. Results are similar to the TCP SYN flood attack experiments.

Overall, without our protection, both the controller and switch consumed around

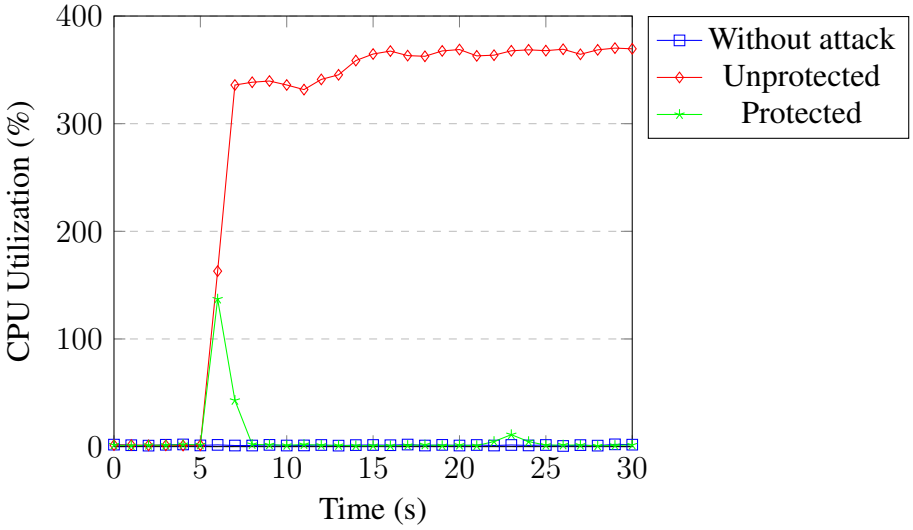

Figure 6.12: Switch CPU under SYN flood.

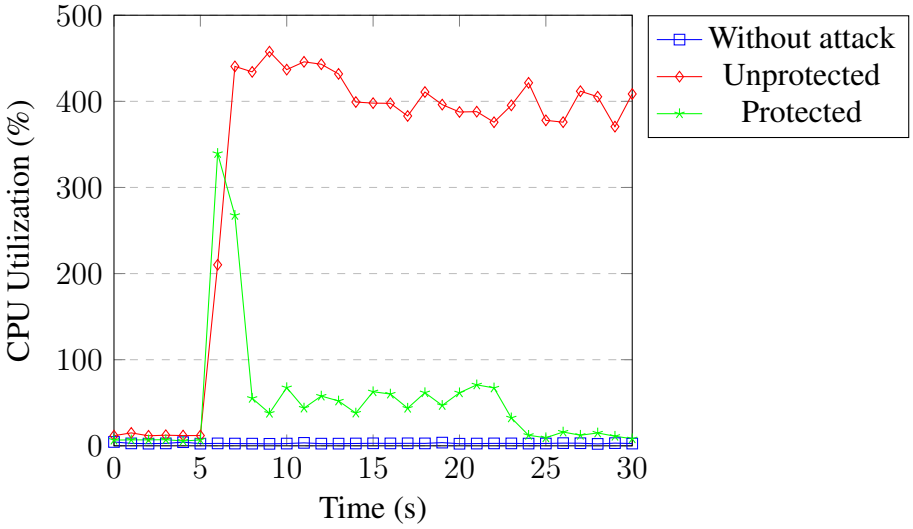

Figure 6.13: Controller CPU under UDP flood.

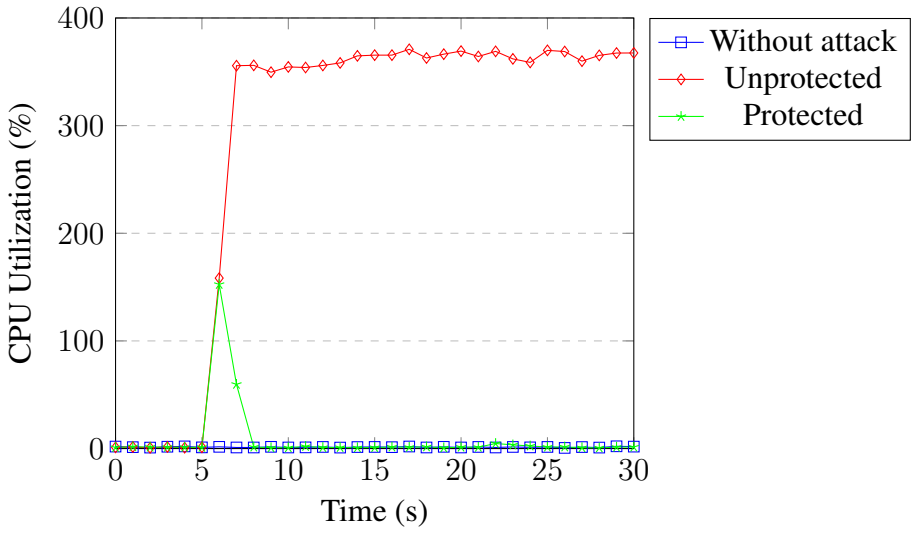

Figure 6.14: Switch CPU under UDP flood.

400% of the CPU, 800% total. DoS attack wasted a huge part of the CPU of the switch and the controller considering maximum CPU utilization of our machine was 1200%. One attacker caused network to use 2/3 of its available CPU. When we performed DDoS attack with 4 attackers, CPU utilization reached to maximum in a short time and SDN controller crashed after some time. When our protection was active, attacks were detected within a second and attackers were blocked immediately and switch's CPU utilization backed to normal, which is close to 2%. All of the packets coming the attacker matched with the block rule and dropped. Our application kept classifying attack rules remained in the switch until they time out. Therefore, CPU utilization of the controller was around 40% for 15-20 second after attack detection. Then, CPU utilization returned back to normal.

CHAPTER 7

CONCLUSIONS

In this work, we proposed an automated, intelligent intrusion detection and mitigation approach for SDN, which aims to provide explainable security in the IoT networks of the 5G era. The proposed approach relies on automated flow feature extraction and highly accurate classification of network flows by a random forest classifier in the SDN application layer, for detecting various classes of attacks and taking remedial action through installation of new flow rules with high priority at the data plane. We presented our SDN-specific dataset modeling a realistic IoT environment, which includes flow data for common network attacks as well as normal traffic, and provided results on the accuracy of intrusion detection as well as performance results in the presence and absence of our proposed security mechanism.

The proposed security approach is promising to achieve real-time, highly accurate detection and mitigation of attacks in SDN-managed networks, which will be in widespread use in the 5G and beyond era. We believe the created dataset will also be a useful resource for further research in ML-based intrusion detection in SDN-managed IoT networks. Our future work will include extension of the created dataset with more attack types and network topologies, as well as evaluation of the proposed security approach with these additional network conditions. We also aim to integrate an interface for interpretability by human experts to further enhance the explainability of the security model.

REFERENCES

[1] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, "Explainable machine learning for scientific insights and discoveries," *IEEE Access*, vol. 8, pp. 42200–42216, 2020.

[2] L. Viganò and D. Magazzeni, "Explainable security," in *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy, September 7-11, 2020*, pp. 293–300, IEEE, 2020.

[3] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *CoRR*, vol. abs/1811.00701, 2018.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[5] H. Mostafaei and M. Menth, "Software-defined wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 119, pp. 42 – 56, 2018.

[6] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.

[7] Q. Long, Y. Chen, H. Zhang, and X. Lei, "Software defined 5g and 6g networks: a survey," *Mobile Networks and Applications*, 2019.

[8] I. F. Akyildiz, P. Wang, and S.-C. Lin, "Softair: A software defined networking architecture for 5g wireless systems," *Computer Networks*, vol. 85, pp. 1 – 18, 2015.

[9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, p. 5–32, Oct. 2001.

[10] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *Journal of Machine Learning Research*, vol. 15, no. 90, pp. 3133–3181, 2014.

[11] S. Rathore and J. H. Park, "Semi-supervised learning based distributed attack detection framework for iot," *Applied Soft Computing*, vol. 72, pp. 79 – 89, 2018.

[12] S. Evmorfos, G. Vlachodimitropoulos, N. Bakalos, and E. Gelenbe, "Neural network architectures for the detection of syn flood attacks in iot systems," Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA'20), (New York, NY, USA), Association for Computing Machinery, 2020.

[13] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine learning-based iot-botnet attack detection with sequential architecture," *Sensors*, vol. 20, p. 4372, Aug 2020.

[14] M. Alqahtani, H. Mathkour, and M. M. Ben Ismail, "Iot botnet attack detection based on optimized extreme gradient boosting and feature selection," *Sensors*, vol. 20, p. 6336, Nov 2020.

[15] N. Ravi and S. M. Shalinie, "Learning-driven detection and mitigation of ddos attack in iot via sdn-cloud architecture," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3559–3570, 2020.

[16] J. Galeano-Brajones, J. Carmona-Murillo, J. F. Valenzuela-Valdés, and F. Luna-Valero, "Detection and mitigation of dos and ddos attacks in iot-based stateful sdn: An experimental approach," *Sensors*, vol. 20, p. 816, Feb 2020.

[17] D. Yin, L. Zhang, and K. Yang, "A ddos attack detection and mitigation with software-defined internet of things framework," *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[18] M. E. Ahmed and H. Kim, "Ddos attack mitigation in internet of things using software defined networking," in *Proceedings of 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 271–276, 2017.

[19] S. S. Bhunia and M. Gurusamy, "Dynamic attack detection and mitigation in iot using sdn," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, 2017.

[20] P. K. Sharma, S. Singh, and J. H. Park, "Opcloudsec: Open cloud software defined wireless network security for the internet of things," *Computer Communications*, vol. 122, pp. 1 – 8, 2018.

[21] P. Bull, R. Austin, E. Popov, M. Sharma, and R. Watson, "Flow based security for iot devices using an sdn gateway," Proceedings of 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 157–163, 2016.

[22] J. Li, Z. Zhao, R. Li, and H. Zhang, "Ai-based two-stage intrusion detection for software defined iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2093–2102, 2019.

[23] P. Amangele, M. J. Reed, M. Al-Naday, N. Thomos, and M. Nowak, "Hierarchical machine learning for iot anomaly detection in sdn," Proceedings of 2019 International Conference on Information Technologies (InfoTech), pp. 1–4, 2019.

[24] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure iot architecture," *Internet of Things*, vol. 3-4, pp. 82–89, 2018.

[25] A. Al Hayajneh, M. Z. A. Bhuiyan, and I. McAndrew, "Improving internet of things (iot) security with software-defined networking (sdn)," *Computers*, vol. 9, p. 8, Feb 2020.

[26] Q. Shafi, A. Basit, S. Qaisar, A. Koay, and I. Welch, "Fog-assisted sdn controlled framework for enduring anomaly detection in an iot network," *IEEE Access*, vol. 6, pp. 73713–73723, 2018.

[27] A. Derhab, M. Guerroumi, A. Gumaei, L. Maglaras, M. A. Ferrag, M. Mukherjee, and F. A. Khan, "Blockchain and random subspace learning-based ids for sdn-enabled industrial iot security," *Sensors*, vol. 19, p. 3119, Jul 2019.

[28] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems," *IEEE Access*, vol. 8, pp. 73127–73141, 2020.

[29] H. Polat and O. Polat, "The effects of dos attacks on odl and pox sdn controllers," in *2017 8th International Conference on Information Technology (ICIT)*, pp. 554–558, 2017.

[30] T. Alharbi, S. Layeghy, and M. Portmann, "Experimental evaluation of the impact of dos attacks in sdn," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, 2017.

[31] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, pp. 77–81, 2015.

[32] C. Gkountis, M. Taha, J. Lloret, and G. Kambourakis, "Lightweight algorithm for protecting sdn controller against ddos attacks," in *2017 10th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 1–6, 2017.

[33] G. Simsek, H. Bostan, A. K. Sarica, E. Sarikaya, A. Keles, P. Angin, H. Alemdar, and E. Onur, "Dropppp: A p4 approach to mitigating dos attacks in sdn," *Information Security Applications Lecture Notes in Computer Science*, p. 55–66, 2020.

[34] KDD Cup 1999. Avaliable online: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (Accessed on 20 November 2020).

[35] 1998 DARPA Intrusion Detection Evaluation Dataset. Available online: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset (Accesses on 20 November 2020).

[36] 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*, pp. 1–6, 2009.

[37] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, p. 357–374, 2012.

[38] Center for Applied Internet Data Analysis. Available online: https://www.caida.org/data/ (Accessed on 20 November 2020).

[39] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network i intrusion detection systems (unsw-nb15 network data set)," Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6., IEEE, 2015.

[40] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," Proceedings of 4th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108–116, January 2018.

[41] A. K. Sarica and P. Angin, "A novel sdn dataset for intrusion detection in iot networks," in *16th International Conference on Network and Service Management, November 2-6, 2020*, IFIP, 2020.

[42] SDN Dataset. Available online: https://github.com/AlperKaan35/SDN-Dataset (Accesses on 12 January 2021).

[43] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[44] ONOS. Available online: https://www.opennetworking.org/onos (Accessed on 20 November 2020).

[45] Open vSwitch. Available online: https://www.openvswitch.org (Accesses on 20 November 2020).

[46] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, p. 115–139, 2006.

[47] hping3. Available online: http://www.hping.org/hping3.html (Accessed on 20 November 2020).

[48] R. K. C. Chang, "Defending against flooding-based distributed denial-of-service attacks: a tutorial," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 42–51, 2002.

[49] A. K. Kaushik, E. S. Pilli, and R. C. Joshi, "Network forensic system for port scanning attack," in *2010 IEEE 2nd International Advance Computing Conference (IACC)*, pp. 310–315, 2010.

[50] Nmap. Available online: https://nmap.org (Accessed on 20 November 2020).

[51] R. Owens and W. Wang, "Non-interactive os fingerprinting through memory de-duplication technique in virtual machines," in *30th IEEE International Performance Computing and Communications Conference*, pp. 1–8, 2011.

[52] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Chapter one - security testing: A survey," vol. 101 of *Advances in Computers*, pp. 1 – 51, Elsevier, 2016.

[53] Boofuzz. Available online: https://boofuzz.readthedocs.io/en/stable (Accessed on 20 November 2020).

[54] iPerf3. Available online: https://iperf.fr (Accessed on 20 November 2020).