REAL-TIME INTRUSION DETECTION SYSTEM IN IOT MEDICAL DEVICES

BY

JOSHUA PHANG JEN HOE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

JUN 2022

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **REAL-TIME INTRUSION DETECTION SYSTEM IN IOT MEDICAL DEVICES**

Academic Session: **JUNE 2022**

I **JOSHUA PHANG JEN HOE**
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible] [illegible]

(Author's signature)	(Supervisor's signature)
Address:	
43, Hala Perajurit 6,	
Taman Kaya,	GAN MING LEE
31400 Ipoh, Perak	Supervisor's name
Date: 7 SEPTEMBER 2022	**Date**: 8/9/2022

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 7 SEPTEMBER 2022

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that ______***Joshua Phang Jen Hoe***__________(ID No: __***18ACB06775***__) has completed this final year project entitled "_*Real-Time Intrusion Detection System in IoT Medical Devices* _" under the supervision of ***Gan Ming Lee*** (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

____
(*Joshua Phang Jen Hoe*)

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Real-Time Intrusion Detection System in IoT Medical Devices**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	Joshua Phang Jen Hoe
Date	:	7 September 2022

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr Gan Ming Lee, for giving me this opportunity to be involved in this project and guiding me in every step of the way as much as possible.

I would also like to express my utmost gratitude to my family members for their love and continuous support throughout my journey as a student.

ABSTRACT

The wide adoption of the Internet of Things (IoT) in the current digital world is gradually increasing with time, focusing on the various benefits and huge convenience IoT can bring about to the way people live. However, new technological advancements will always be introduced to potentially new, unknown security threats and vulnerabilities, hence a real-time intrusion detection system is implemented in this project. This research-based cybersecurity project highlights the importance of an intrusion detection system in improving the security level of the IoT medical devices. The design of the real-time IDS revolves around setting up simple IoT devices resembling IoT medical devices to form an IoT network, performing attacks on the network, capturing network packets in real-time, and classifying network data with a deep learning framework to help in identifying modern intrusions and network traffic anomalies. Some network attacks are performed within the network and the packet data are captured at the same time. Generative adversarial network will be used as the deep-learning-based generative model for anomalous intrusion detection purposes. The model itself will be trained and tested with a network intrusion dataset for benchmarking the model performance. In the context of real-time IDS, this project aims to improve the security aspects of the IoT medical devices, and possibly spark the importance of security technologies like IDS in the IoT industry.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoMT	Internet of Medical Things
IoT	Internet of Things
IDS	Intrusion Detection System
ICMP	Internet Control Messaging Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
MITM	Man-in-the-Middle
DDoS	Distributed Denial of Service
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Network
PIR	Passive infrared
ML	Machine Learning
GAN	Generative Adversarial Network
IDE	Integrated Development Environment
OHE	One-Hot Encoding
CSV	Comma-separated Values
ReLU	Rectified Linear Unit
API	Application Programming Interface
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
gRPC	gRPC Remote Procedure Call
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Chapter 1
Introduction

1.1 Problem Statement and Motivation

Medical devices that are connected to the Internet have been significant in shaping the way healthcare entities treat patients with advanced technologies embedded into such devices. In the current digital era, patients are able to receive treatments and track their health status in real-time due to the wide adoption of IoT medical devices in the healthcare industry. However, there has been a surging number of new challenges and issues pertaining to the increasing demands of IoT medical devices or Internet of Medical Things (IoMT). There is a wide range of varieties for all kinds of IoT medical devices to be used in hospitals, healthcare clinics and even general IoMT for personal use with some examples like smartwatches that monitor heart rate and pulses. Among medical devices in hospitals and clinics in the context of a larger scale are remote patient monitoring, connected inhalers, ingestible sensors, robotic surgery equipment and many more [1]. However, the current phase that the healthcare industry is residing in has exacerbated the problems IoMT encountered from the security standpoint and further increased the demand of IoT medical devices since the start of the COVID-19 worldwide pandemic. There are numerous IoMT that are being vulnerable and exposed to various security threats and attack vectors in many hospitals and healthcare centres around the globe. Today, IoMT are one of the most sought-after attack targets for unethical parties and cybercriminals alike for apparent reasons like monetary gains and social propaganda. In addition, the number of preventive measures taken to safeguard and protect such medical devices and systems from vulnerabilities and exploits is not enough to combat the severe issues of medical devices being compromised on a day-to-day basis [2].

Since the major proportion of the IoT medical devices is optimised and used in hospitals and healthcare clinics, there is a huge reliance on these medical devices that have the ability to gather and store sensitive information about patients, particularly personal identifiable information (PII) which includes a patient's full name, address, contact details and many more. With the vast amount of security breaches and cyber-attacks in place, the affected IoT medical devices face a high risk of data exposure to the public regarding patients' personal data, as well as healthcare organisations' information. Other than that, IoT medical devices with critical functionalities that are affected by cyber-attacks might result in life-threatening consequences whereby patients who heavily depend on these types of medical devices may lose their lives.

Therefore, there is a need to design a real-time intrusion detection system in every network architecture of all healthcare organisations and entities to ensure the safe use and normal operational behaviour of medical devices in real-time. The security robustness and reliability of IoT medical devices have to be improved by implementing intrusion detection systems as an extra security solution to detect known and unknown attacks, as well as to protect the patients' and healthcare organisations' data and privacy.

1.2 Objectives

The aim of the thesis is to propose a real-time IDS that is able to detect any possible network intrusions in IoT medical devices. The effects of exploited vulnerabilities in IoT medical devices of all operations result in many undesirable consequences for patients and healthcare industries. Such occurrences are usually the result of the lackadaisical of IDS in IoT medical devices or networking components in these devices. In this thesis, the presence of an IDS in any IoT devices will gear towards the efforts in securing as many IoT medical devices as possible and reducing the security risks of such devices. The proposed real-time IDS will be designed to be able to identify modern network intrusion and abnormal network traffic patterns through a deep learning framework, as well as to provide a viable solution for anomaly-based intrusions in general.

1.3 Project Scope

The scopes of this project include exploring certain vulnerabilities that are presented in IoT medical devices and proposing an effective real-time IDS based on existing and unknown vulnerabilities. Highlighted vulnerabilities will be investigated and ventured to understand the base concepts towards the foundation of the vulnerabilities that will lead to cyber-attacks towards IoT medical devices in general. A network-based IDS is to be proposed for detecting different vulnerabilities as an additional countermeasure to security problems in IoT medical devices. The IoT medical devices to be used for measuring the effectiveness and efficiency of the proposed IDS could be sensor nodes or tiny computer systems that are able to mimic the representation of such medical devices. Any information regarding the potential vulnerabilities that might exist in the replica sensor nodes is useful towards the actions and decisions to be made in the process of simulating an IDS in real world contexts.

1.4 Contributions

The main prospect of this project is to propose an IDS that can safeguard IoT medical devices in real-time. With the lack of sufficient security provision in many of IoT medical devices today, the proposed real-time IDS will offer an extra layer of security measure to further solidify the security aspect of such devices. This will help the healthcare industry to propel towards the importance of security in the use of IoT medical devices, as well as to ensure the confidentiality and integrity of medical data, and the functionalities of such medical devices to be protected from exploits and vulnerabilities.

1.5 Report Organisation

This report is organised into 7 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Model, Chapter 4 System Design, Chapter 5 Experiment/Simulation, Chapter 6 System Evaluation and Discussion, Chapter 7 Conclusion and Recommendation. The first chapter covers the brief introduction on IoT medical devices and problem statements, as well as the project objectives and the scope to be covered. The second chapter is the literature review regarding some of the technologies related to the proposed real-time IDS, as well as the previous research works on the IDS domain. The third chapter covers the system model of the proposed IDS in a top-level perspective, detailing the general flow of the implementation. The fourth chapter is about the system design of the proposed approach, where the IDS implementation will be shown in a more detailed manner, as well as to describe the components of the system from a block diagram. The fifth chapter entails the experiment or simulation of the proposed real-time IDS, where details like set-ups, configurations and system operation will be covered. The sixth chapter is about the evaluation and discussion of the system, where the testing metrics and the results are shown and explained. The final chapter covers the conclusion part of the report that will include the final remarks of this project and some recommendations for any future potential improvements.

Chapter 2
Literature Review

2.1 Review of the Technologies

In this section, some common vulnerabilities that occur in IoT medical devices will be covered, as well as some relevant technologies to be used in the implementation and previous works on IDS in a brief manner.

2.1.1 Vulnerabilities in IoT Medical Devices

Since the introduction of IoT medical devices in the medical field, there is a range of new and existing threats and vulnerabilities to be presented in almost every of these medical devices, no matter the older or newer versions of such devices and its embedded software. Many IoT medical devices are prone to different kinds of attack vectors and there will always be new, unknown vulnerabilities to be discovered, also known as zero-day exploits. Such exploits are generally difficult to detect and be made known to security experts or researchers alike as these exploits could pose a damaging risk to patients and healthcare organisations. Other than that, such medical devices can also be susceptible to indirect attacks in which infected hosts or endpoints in a network may enable other forms of attacks like malicious code injection, resulting in IoT medical devices of the same network to be compromised [3].

Vulnerabilities of varied complexity levels are generally subjected to the purpose of attackers performing various kinds of cyber-attacks to target specific IoT medical devices. Although there are no defined set of rules and procedures in a typical planning of a cyber-attack, some vulnerabilities that target the weaknesses of a certain aspect of security properties are listed as follows:

2.1.1.1 Denial of Service (DoS) attacks

DoS attacks remain one of the most common vulnerabilities towards the implementation of IoT medical devices. This can be seen where a network with many interconnected IoT medical devices and systems in a healthcare setting can be overwhelmed with forged traffic requests from attackers' machines. Other than that, the adoption of cloud servers for all kinds of IoT medical devices will also introduce new threat landscapes and evolved methodologies of DoS attacks. Although some hospitals or clinics in the healthcare industry in which most private networks and cloud servers have several security measures and tools to strengthen the security aspects of IoT medical devices, this is still not adequate in curbing prominent DoS attacks. As there are no security strategies that can fully prevent all kinds of DoS attacks, attackers will always try to look for new DoS attacks or even revamp existing attacks to pose negative effects on IoT medical devices and the network architectures.

Some possible variants of DoS attacks in IoT medical devices include ICMP flooding attack and SYN flooding attack. Adversaries could also combine multiple forms of DoS attacks and transform them into multiple computer systems infecting a single target, also known as Distributed Denial of Service (DDoS) attacks. Internet Control Messaging Protocol (ICMP) flooding attack focuses on overloading a targeted medical device with ICMP echo packets, also known as pings in a more general term. Since the main function of an ICMP ping is to test the connectivity of two hosts or endpoints with echo requests and echo replies, there is no mechanism of determining and identifying the authenticity of the pings. Given that the targeted medical device in a network will respond with an equal number of reply packets, packets can be forged using tools like *Scapy* and *hping* and be used to flood the target or network devices like routers.

SYN flooding attacks apply a similar concept compared to ICMP flooding attacks in which the idea of overloading the targeted medical devices leads to disrupting and halting the functionalities of such medical devices. However, SYN flooding attacks target IoT medical devices of more importance in transmitting medical data to healthcare servers using Transmission Control Protocol (TCP) in the transport layer of the TCP/IP networking model. Those servers that are very dependent on the data provided by medical devices to provide crucial web services for healthcare personnel and patients alike are the primary target for attackers. This kind of attack abuses the three-way handshake process for a TCP connection as multiple SYN packets are sent to the targeted server and prevent any TCP packets with RST flag enabled from being generated. If successful, the bandwidth and memory resources of the targeted server will be depleted and might affect its operation, as well as further attacks can be carried out due to incomplete TCP connections.

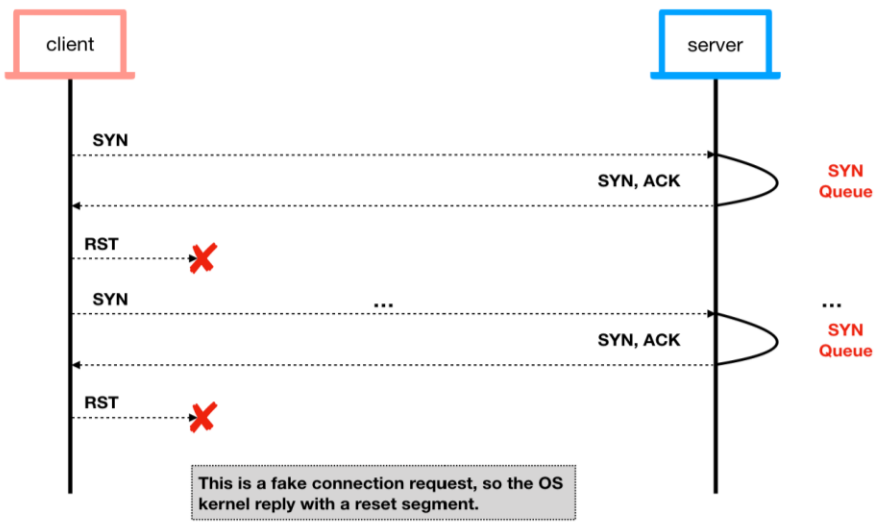

client
server
SYN
SYN, ACK
SYN Queue
RST
SYN
...
SYN, ACK
...
SYN Queue
RST
This is a fake connection request, so the OS kernel reply with a reset segment.

Figure 2.1: SYN flood attack

2.1.1.2 Man-in-the-middle (MITM) attacks

IoT medical devices function by making use of wireless sensor communications, in which this might be vulnerable to man-in-the-middle attacks, jeopardising the confidentiality, integrity, and availability of such medical devices [4]. A MITM attack occurs when an attacker monitors and tampers with communications between two legitimate parties, making changes to the transmitted data in the process of doing so. This is comparable to a typical eavesdropping attempt, in which the attacker listens in on the discussions of two individuals. MITM attacks include network content spoofing, network eavesdropping, session hijacking and IP spoofing.

MITM attacks have evolved over the course of IoT medical devices adoption in the healthcare industry. Generally, there are two types of such attack: passive or active MITM. A passive attack is all about intercepting and reading the messages that are exchanged between two entities without tampering the communication aspect. Passive MITM attacks are initially carried out in medical devices to gather as much information about the functionalities of the targeted medical devices as possible, as well as the nature of the data flow between the two communicating entities. This would normally be the first step for attackers to understand how the medical devices interlink with one another and the nature of the network. Active MITM attacks are then followed up after performing passive attacks whereby the attackers are able to gain access to targeted medical devices and manipulate or modify the transmitted data to operating servers. When parts of the functionalities of a medical device are compromised due to the attacker's control, things would start to go haywire. Some undesirable scenarios like falsifying communications between a medical device and a server and disabling the core functionalities of a medical device would result in the endangerment of patients' lives and the integrity of doctors or nurses that are highly responsible for the treatment of patients. When both passive and active MITM attacks are carried out simultaneously, such actions may lead to other deadly attacks that will result in further complicated problems and scenarios, causing the process of neutralising and mitigating cyber-attacks by security vendors and experts to be more time-consuming and troublesome.

2.1.1.3 Side channel attacks

Side channel attacks involve the exploitation of observable physical properties emitted from the physical components of IoT medical devices. This form of attack focuses more on accessing sensor components found in IoT medical devices, such as gyroscope and accelerometer, that transmits data from the hardware aspects of the sensors. These sensors can be exploited in which the transmitted data to a remote server is intercepted, hence leaking private information about the medical devices and those belonging to users [5]. Other sensitive data emitted from environmental and electrical properties from internet-enabled medical devices like cryptographic key implementation and user inputs like passwords and PINs can be disclosed from side channel attacks as well [6].

There are a few components of side channel attacks that are responsible for the exploitation of physical components in IoT medical devices to perform cryptographic key recoveries and data extraction. Power analysis is one of the side channel attack components whereby the differences in power consumption of medical devices are monitored and differentiated to determine voltage fluctuations. Power analysis can be further broken down into two types: simple power analysis (SPA) that identifies the instructions and cryptographic operations under execution using waveforms, and differential power analysis (DPA) in which secret keys are traced and discovered from ciphertexts using advanced statistical analysis [6]. Other analysis components like electromagnetic emission (EM) analysis, acoustic cryptanalysis and temperature analysis exploit a certain physics or environmental aspect used by a certain hardware sensor in IoT medical devices. The general idea behind side channel attacks is to discover the differences or redundancies in each sensor component in medical devices, so that sensitive information regarding the device, user or embedded system within the device itself can be extracted and captured. Such information can be used for several other purposes like exploiting the hardware vulnerabilities of medical devices and performing other cyber-attacks based on the personal information of users.

2.1.2 UNSW-NB15 Dataset

The UNSW-NB15 dataset is a network intrusion dataset that contains a mixture of real modern network activities and synthetic attack scenarios. It has nine type of different network attacks and was constructed from the IXIA PerfectStorm traffic generator tool configured by researchers in UNSW Canberra, along with tcpdump tool to capture 100 GB of raw traffic. Argus and Bro-IDS, now known as Zeek, were also used to extract the original 49 features in the dataset [10].

Regarding the dataset availability, many find it difficult to look for a reliable network intrusion dataset due to the scarcity of public datasets. Therefore, the creation of UNSW-NB15 dataset would be a more significant dataset in detecting novel attacks compared to existing benchmark datasets like KDD99 and NSL-KDD datasets [11]. Since the creation of UNSW-NB15 dataset, it was used in an increasing number of areas of interest fields and was proven to be comparatively good in evaluating the performance of the proposed frameworks in the intrusion detection/anomaly detection systems [12] [13]. In addition, some new datasets with modified features were created from the UNSW-NB15 dataset to cater for more specific use cases, with ML-based network IDS as an example [14].

The UNSW-NB15 dataset has a total number of 2.5 million records separated in four CSV (Comma-Separated Values) files. It also contains a training dataset and a testing dataset partitioned from the full version of the dataset, making it suitable and adequate to train ML algorithms.

2.1.3 Packet Sniffing

Packet sniffing is a method to analyse and intercept network packets in a network. Packet sniffing is usually performed with network protocol analysing tools to capture data travelling across a network, allowing those tools to capture data packets in a very fast manner. Packet sniffing tools are able to capture and read packets at the network layer of the TCP/IP layer, along with providing important packet features within the captured packets like packet length and protocol types. It is normally done to monitor network traffic by network administrators to ensure the network traffic is not malicious. It is also used to assess network performance, extract features from network traffic and identify potential network intrusions through packet analysis [16].

2.2 Review of Previous Research Works on IDS

Hodo et al. [7] proposed the implementation of Artificial Neural Network (ANN) as one form of network-based intrusion detection system. The proposed ANN model in IDS was tested and trained with a simulated IoT network with interconnected sensor nodes as IoT devices. Hodo et al. [7] also proposed the threat analysis of IoT using ANN model to monitor and identify the network specifically for DoS/DDoS attacks. A multi-level perceptron, which is a type of supervised ANN, is trained using internet packet traces and is tested according to its ability to thwart DDoS/DoS attacks. The experimental architecture used in this proposed research is made of five node sensors, in which one of them acts as a server and is set as the primary target for attackers. Their results of simulated DDoS attacks with the accumulation of more than ten million packets were based on the detection of normal and threat traffic patterns, making this proposed ANN model as an anomaly-based type IDS for DDoS/DoS attacks in the network layer.

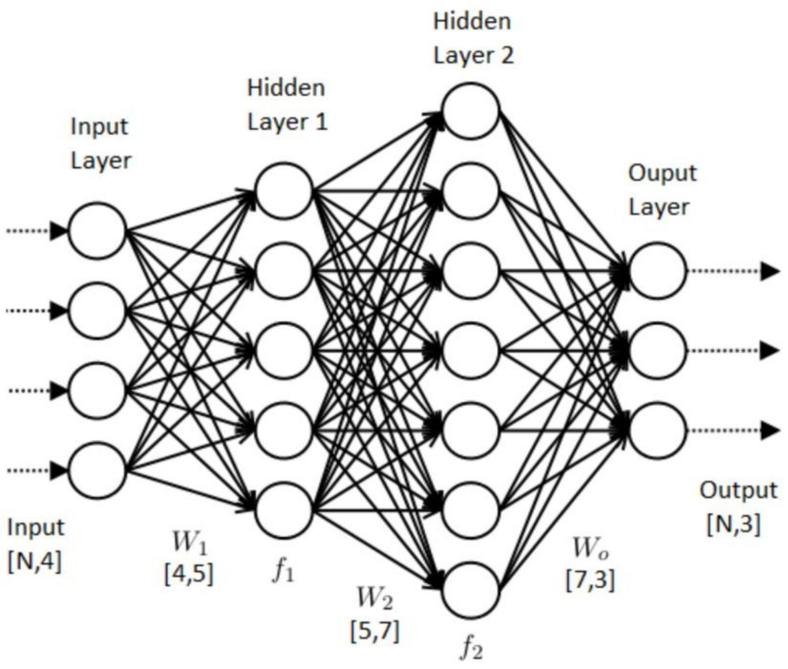


Figure 2.2 Artificial neural network

Soe et al. [8] proposed a machine learning-based botnet attack detection IDS framework with sequential detection architecture. Their approach of detecting botnet attacks was based on three different machine learning algorithms, including ANN, J48 decision tree, and Naive Bayes. Their proposal of the sequential attack detection architecture revolved around two phases, in which the first phase performs building training models and collecting data, and the second phase detects botnet attacks from incoming network traffic based on the traffic patterns analysed by their end-product IDS engine.

Cervantes et al. [9] proposed an IDS to identify sinkhole attacks on 6LoWPAN systems for the IoT in general. Their proposed IDS is a hybrid IDS that can detect anomaly-based and signature-based network activities on the routing services. Their approach combines several strategies for detection of attackers by analysing the behaviour of the sensor nodes. Their proposed IDS also establishes dynamic clustering to support IoT data transmission by reputation and trust mechanisms. Based on their simulation results, the sinkhole detection rate in their proposed IDS can achieve up to 92% in a fixed scenario and 75% in a mobile scenario.

Chapter 3

System Model

This chapter covers the system model at a top-level design, detailing the major components of the system on a surface level. A use case diagram and an activity diagram will also be included to briefly illustrate the use of the IDS.

3.1 System Architecture Diagram

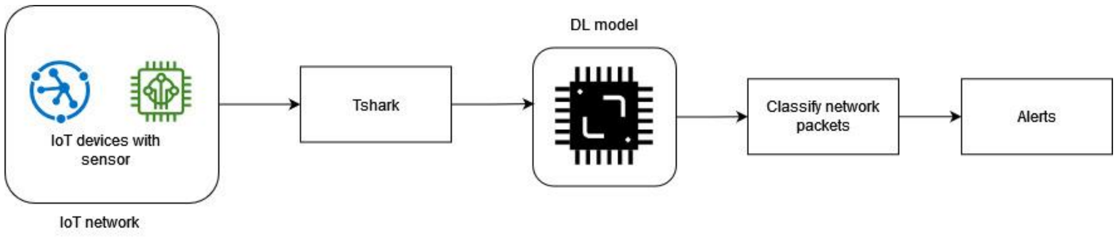


Figure 3.1 Top-level system architecture diagram

The system architecture of the IDS involves few main stages: training the ML model, packet sniffing with Tshark and classifying network packets. The first stage involves training the model with a deep learning method approach, utilising the GAN architecture where algorithms like backpropagation and feature calculations are used to continuously improve the model throughout the training process. The following stage is the packet sniffing process using a network protocol analyser tool called Tshark. This process will be done under a running model server where it hosts the built model file that is responsible for the logic of classifying network packets and captures network packets with the appropriate filters to obtain the relevant ones. The last stage involves classifying captured network packets, where the prediction functions in the model server will classify the packets based on the trained model file and classify them accordingly.

3.2 Use Case Diagram and Description

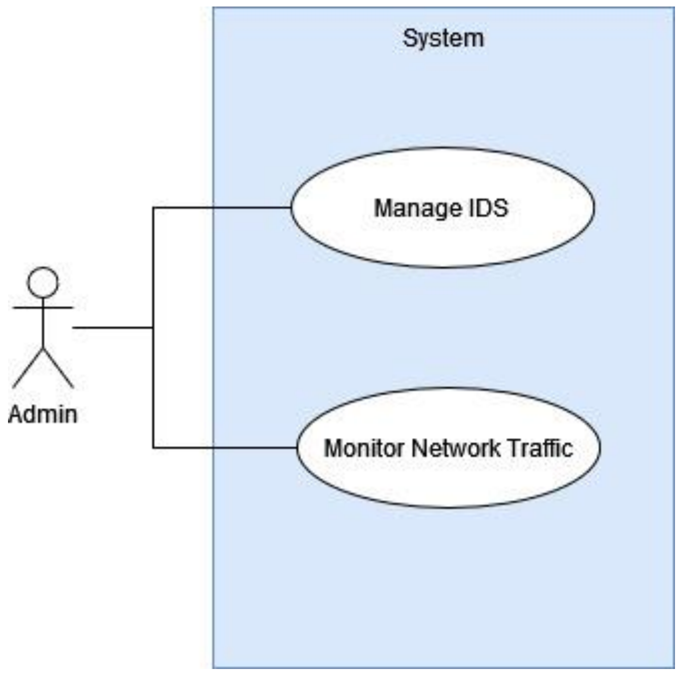


Figure 3.2 Use case diagram

The use case scenario for the real-time IDS is applicable to everyone who requires a system to monitor network traffic in a private network. This may be applicable for users or organisations who operate many IoT devices in a network, including internet-enabled medical devices. A given example can be an admin in an organisation handling all the monitoring and the management of the network traffic and the IDS itself, respectively. The admin will be able to configure and modify the command for packet sniffing in the model server with the appropriate filters for specific use cases, like monitoring only TCP and UDP traffic in a network. By doing this, the admin can monitor the filtered network traffic for any possible network attack occurrence, and also be notified of network attacks if any.

3.3 Activity Diagram

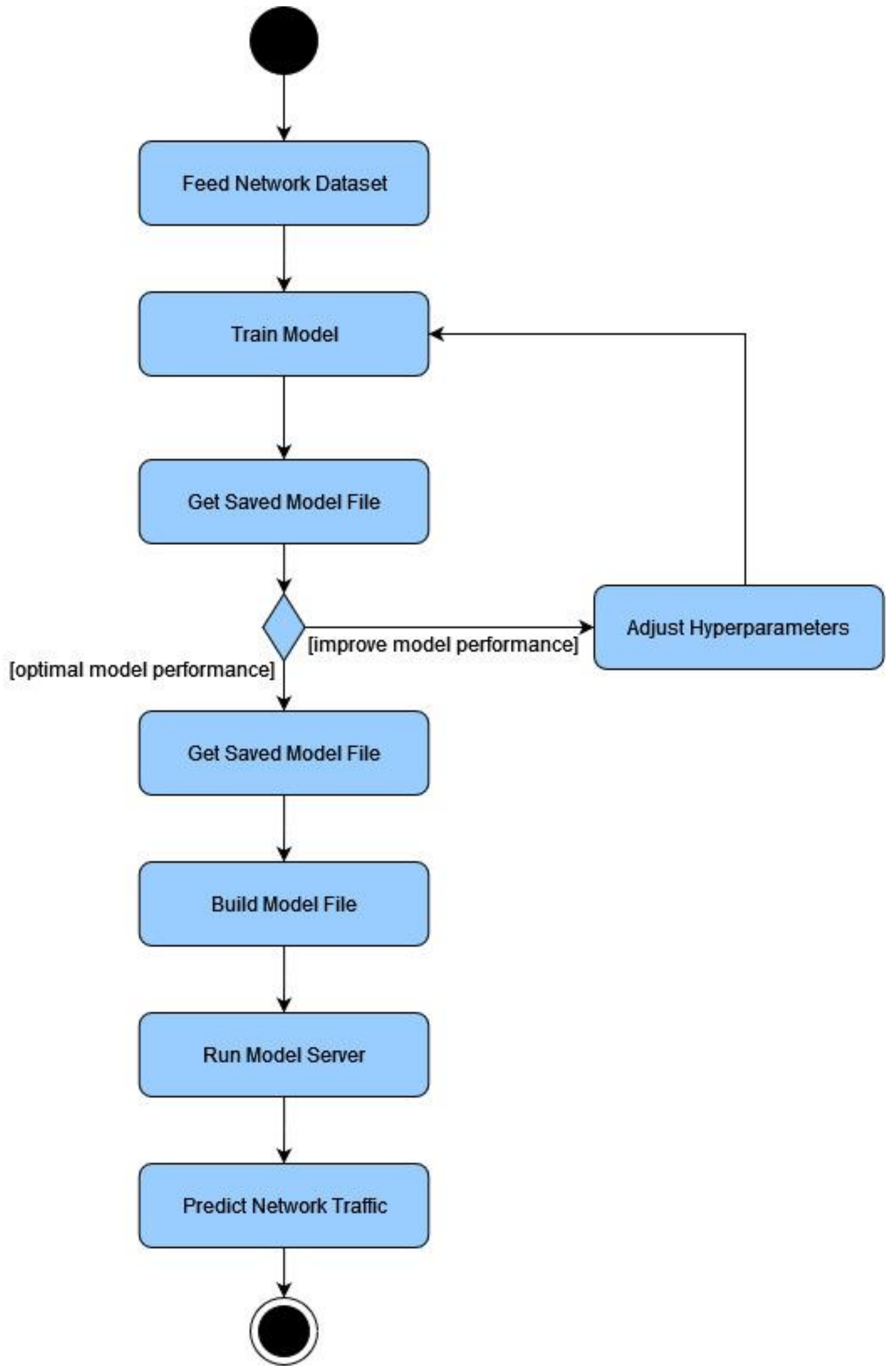


Figure 3.3 Activity diagram

The activity diagram in Figure 3.3 shows the general flow of implementing the real-time IDS. The first few processes involve training the ML model by feeding the pre-processed dataset into the model and obtaining the saved model file afterwards. The model can be improved by adjusting the necessary hyperparameters that might affect the performance of the model with every slight adjustment. Once the model is tested and is satisfactory, the trained model is built and compiled into a model file which is specifically used for running it with a model server. Combined with some client prediction software packages and necessary libraries, the model server will be ready to be used as a real-time IDS to classify network traffic.

Chapter 4

System Design

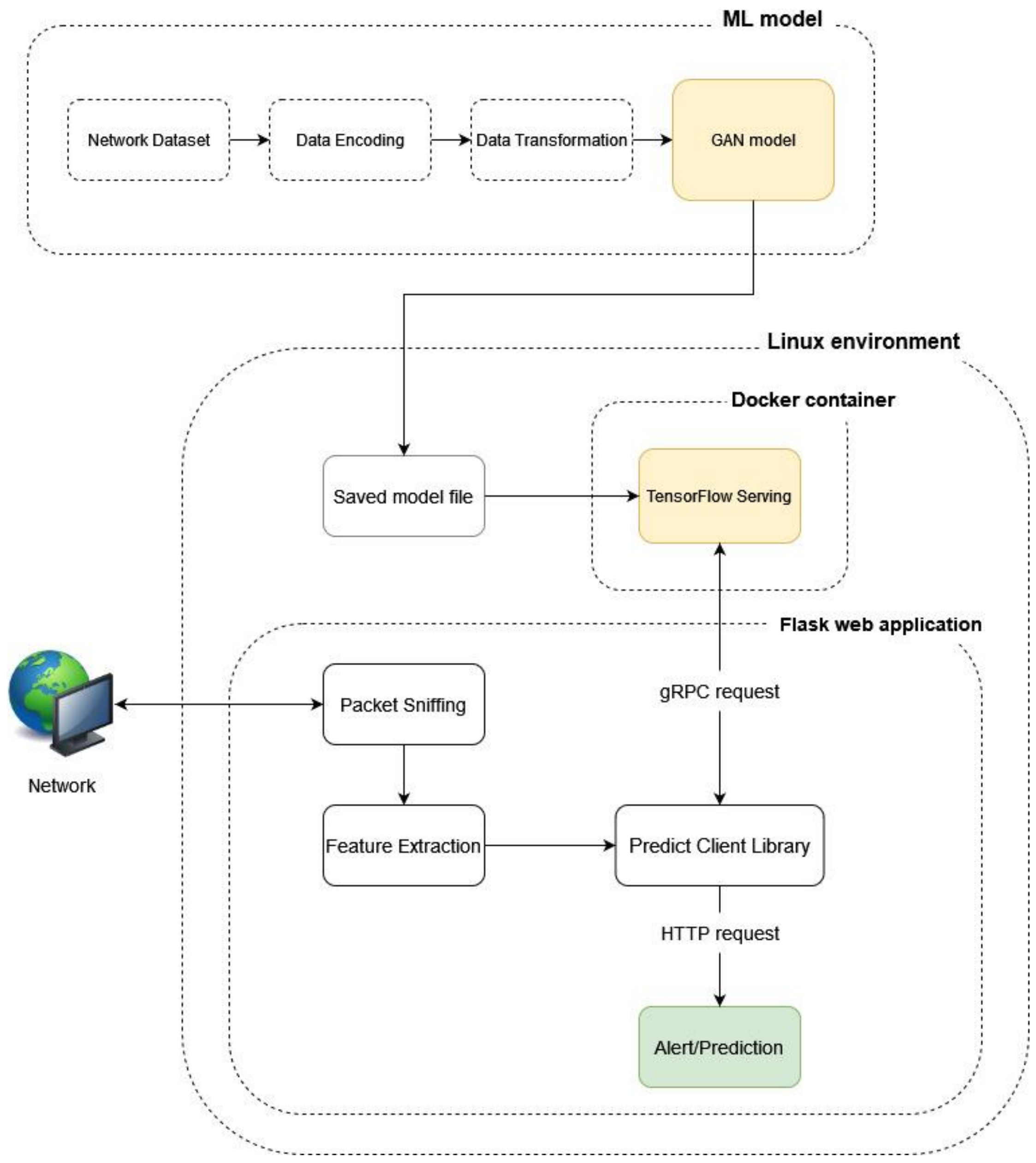


Figure 4.1 System block diagram

The implementation of this research-based project is categorised into five methodological steps: data pre-processing, GAN implementation, network attack simulation, packet sniffing and model deployment.

4.1 Data Pre-processing

Data pre-processing is the action of modifying the structure and nature of the input data for subsequent training of the ML model. It is typically the initial step that should be performed before feeding the data into the ML model. The tasks involved in this data pre-processing step are feature selection, categorical data encoding and data transformation.

4.1.1. Feature Selection

The training of the proposed ML model relied on the use of UNSW-NB15 dataset, originally consisting of 49 attributes in total. The features in the dataset were curated based on thorough analysis on the network attack simulations performed and the use of different algorithms and tools to extract those 49 features in the dataset itself. However, to ease the process of packet sniffing and due to the limitations of the packet sniffing method used, the features in UNSW-NB15 dataset were reduced to three instead of 49 of them. The selected features shown in the table below constitute the main features of a network packet that could potentially differentiate whether the network traffic is normal or malicious in nature.

Table 4.1 Selected features from UNSW-NB15 dataset

No.	Selected feature name
1	proto
2	sbytes
3	sttl

4.1.2 Categorical Data Encoding

One-hot encoding (OHE) was used to perform categorical data encoding since most ML models can only take in features that are numerical in values instead of nominal values. In a typical ML model, nominal categorical data are encoded using ordinal encoding where there are known correlation between features and the order of data in each feature causes the ML model to assign importance to the ordinal nature of data. However, in this project, OHE is preferred compared to ordinal encoding due to the categorical features in the UNSW-NB15 dataset not belonging to any form of ordered relationships between them. The use of OHE in categorical data encoding creates a new column for each unique category value, resulting in additional columns. Therefore, categories with highest number of frequencies are only selected to reduce the number of features.

4.1.3 Data Transformation

After performing manual feature selection and categorical data encoding on the input data in UNSW-NB15 dataset, all the data in the training and testing UNSW-NB15 datasets were converted to NumPy array datatype. This was done to allow the transformed data to fit into the ML model for training and testing purposes. Since the ML model made use of TensorFlow libraries and codes, data transformation was required in order to allow the ML model to process the data in both training and testing datasets.

4.2 GAN Implementation

The implementation of the real-time IDS was based on GAN as the proposed ML model. The original source codes for GAN were from Zander Blasingame and can be found on GitHub [18]. The obtained GAN source codes were used solely for experimenting the usefulness of GAN as the ML model for testing the effectiveness of an IDS. The modified version of the source codes included some changes on the source codes regarding the data pre-processing part, as well as changes on the compatibility issues of some libraries and version support.

The general architecture of GAN involves a deep learning framework which is based on the concepts of a generative model. The GAN architecture has two sub-models: a generator model that is responsible for generating new examples from the original ones and a discriminator model for differentiating real examples from the fake ones, and vice versa. The generator model learns to generate fake data with some random input from the real data based on the feedback from the discriminator and aims to make the discriminator model classify its output as real data. The discriminator model then tries to differentiate real data from the generated data and aims to correctly identify real data from the fake ones.

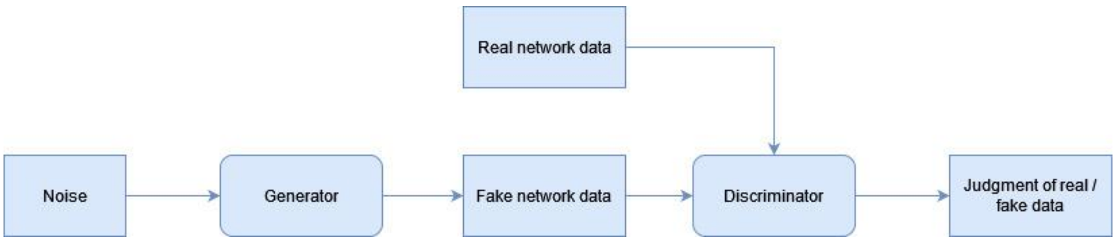


Figure 4.1 A typical generative adversarial network

In the GAN model, several hyperparameters were included as part of the training process. Some of the important hyperparameters in this model are the number of hidden layers, the number of features, learning rates for discriminator and generator models, batch size and the number of epochs. All the hyperparameters involved in the model have a direct impact on the performance of the model and should be adjusted accordingly every time the model is trained for optimal performance.

In the input layer and each hidden layer of the model, all the input for these layers were activated using Leaky ReLU (Rectified Linear Unit) function. The Leaky ReLU activation function is a piecewise linear function, in which the positive input will remain positive output, otherwise the output will be zero. However, Leaky ReLU is slightly different compared to the normal ReLU function, where the Leaky ReLU has an extra parameter called *alpha*, that will covert near-zero positive values to 0 and allow small negative values. The equation below depicts the equation of the Leaky ReLU function:

$$Leaky\ ReLU\ (x) = \max\ (alpha\ \times x, x)$$

The output for the generator model was activated using hyperbolic tangent (tanh) function, whereas the output for the discriminator model was activated using the Sigmoid function. The tanh function maps the generated data in the generator model to the range of (-1,1) for the fake discriminator model, while the real discriminator model generates the output of Sigmoid function, which is the result of the prediction in the form of probability scores and the determinant of the classification outcome. The equation of the Sigmoid function is depicted below:

$$Sigmoid\ (x) = \frac{1}{1 + e^{-x}}$$

Based on both generator and discriminator models, there will be some errors when it comes to training models. The error, also known as loss, is a penalty for a bad prediction based on the input data. It indicates how far the predicted value is compared to the gradient value, during which a loss function is needed to estimate the loss of the model. In this case, the sparse cross entropy was used as the loss function for both sub-models, which the weights and bias of the model are updated in each training epoch to minimise the loss on subsequent model training. Since the end output of the model training produces probability values, sparse cross entropy loss function was preferred compared to the binary cross entropy. As for the optimiser in the model, Adam optimiser was used in both generator and discriminator models with a learning rate of 0.004 and 0.002, respectively. After each training epoch, backpropagation will occur where both the generator and discriminator neural networks have the weights and the biases adjusted from the previous accumulated calculations of all neurons in all layers. This is done to minimise the losses for both generator and discriminator networks and potentially increase the model performance.

4.3 Network Attack Simulation

To test the performance of the trained model, network attack simulation had to be performed on the targeted device. Network attacks like port scan attack and DoS attacks were conducted as an experiment for testing the real-time network classification in the IDS. Some attack tools like hping3 and Nmap can be used to simulate these kinds of attacks, performing attack simulations on the targeted IoT device. Such tools are useful in analysing how the attacks are done and the details behind the idea of the attack, giving insights on the concepts of each attack type and ways to potentially prevent them.

4.4 Packet Sniffing

The packet sniffing method was used to extract the necessary features from a captured network packet for the testing of ML-trained IDS. Tshark [15] was used as the packet sniffing method for feature extraction. Tshark is a command-line network traffic analyser tool that allows packet data capturing from a live network or packet reading from a saved capture file. Unlike other network traffic analyser tools like Wireshark, Tshark provides more flexibility in terms of processing information and extracting custom features from captured packets in a live network. It was used to extract the few network features needed for feeding the processed captured packets into the ML-based IDS for testing purposes.

Tshark works alike with tcpdump, another popular network analysis command-line tool, which uses the pcap library to capture network traffic from an available network interface and displays the standard summary output for each captured network packet. Tshark is generally preferred over GUI-enabled network analyser applications like Wireshark due to its ability of capturing network packets based on certain set of filters and displaying only the relevant packet fields for network analysis. Tshark is built with various options, also known as flags, which allow the modification of the outputs based on the given filters. On top of that, the captured network packets can be decoded in data formats like JavaScript Object Notation (JSON), pcap files and CSV files for further processing and analysis.

To feed captured packets into the trained ML model, main fields like -T flag and -e flag were included in the constructed Tshark command. The -T flag was used to specify a format of the output for the decoded packet data, whereas the -e flag was responsible for adding a field to a list of fields from a captured packet. The -f flag specified the protocols of the captured packets to be displayed, in this case there would be only packets with TCP or UDP protocol as the transport layer in the TCP/IP model being filtered out as the desired output. All these flags were combined in a Tshark command to capture any network packets in a live network interface and decode the packets into JSON format as the output for feeding the decoded data into the trained ML model.

4.5 Model Deployment

The trained ML model for the IDS has to be deployed to be able to detect possible network attacks in real-time. In this case, TensorFlow Serving [16] was utilised to make use of the ability of the ML model to predict whether the incoming network traffic is normal or malicious. TensorFlow Serving is a flexible, high-performance serving system for ML models that are designed for development and production environments. Most ML models that are built using TensorFlow and Keras API software libraries are well suited for the use of TensorFlow Serving in actual ML model deployment. To facilitate the use of TensorFlow Serving easily, a Docker image containing all the necessary libraries and components of TensorFlow Serving was used. Docker simplifies the use of ready-made software products and provides portability in the form of containers. The Docker image built with TensorFlow Serving modules was then containerised in a virtual container and the container was activated to start serving the trained ML model.

The backend of the real-time prediction system in the form of Docker application was supported with Flask [17]. Flask is a simple, yet useful microframework written in Python for web development purposes. Flask can be built in the form of a web application without any specific external tools and libraries, making it an efficient web framework for actual web application deployment.

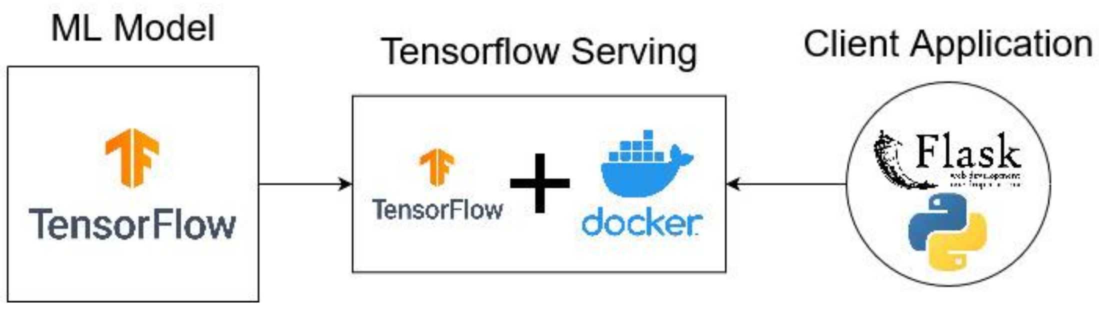


Figure 4.1 Model Deployment with TensorFlow Serving and Flask

The final step to the completion of deploying the ML model into production use is to build a Flask application for the ML model under the TensorFlow Serving system to identify possible network intrusions. The TensorFlow Serving docker image was first retrieved from the official TensorFlow docker account, followed by starting a Docker container which would open the specified REST API port and execute the saved ML model file from the training of the model itself.

Chapter 5
Experiment/Simulation

5.1 Hardware Setup

In this project, there were a few hardware equipment involved in conducting the experimental run of the ML-based IDS. To train the ML model, any computer with decent specifications, specifically for processor and available memory space, would be sufficient. In this case, a desktop was used as the main host machine to handle all the training of the ML model and hosting of the Linux-based virtual machine for real-time IDS testing.

Table 5.1 Specifications of desktop

Description	Specification
Processor	Intel Core i5-10400F @ 2.90GHz, 12 Cores
Operating System	Windows 10
Graphic Processing Unit	NVIDIA GeForce RTX 2060
Memory	16GB DDR4

For the training of the ML model, TensorFlow libraries require the use of GPU by default as a pre-requisite for the ML model training process. However, TensorFlow libraries also support the use of CPU in training a model, and the comparison between CPU and GPU in terms of model training performance is negligible. Since the ML model training process only involved a network traffic dataset consisting of just text-based data rather than graphical or visual data like images and videos, either of them can be used with no adverse differences in performance. In the case where there are no compatible GPUs supported by TensorFlow libraries to be found in the computer system, any available CPU will be used for model training instead.

Table 5.2 Specifications of Linux virtual machine

Description	Specification
Processor	Intel Core i5-10400F @ 2.90GHz, 3 out of 12 Cores used
Operating System	Linux
OS Distribution	Ubuntu-based Kali Linux Rolling
Memory	3GB

As for starting up a Linux-based virtual machine, the specifications listed in Table 5.2 were more than sufficient for the real-time prediction of network traffic. The allocation of the number of processor cores and the memory space can be at least two each respectively, to keep the Linux VM up and running in stable conditions. However, more allocation of these computing resources is recommended as the minimum specification requirements for more compute-intensive Linux distributions like Kali Linux increase, depending on the released distribution version and its usage.

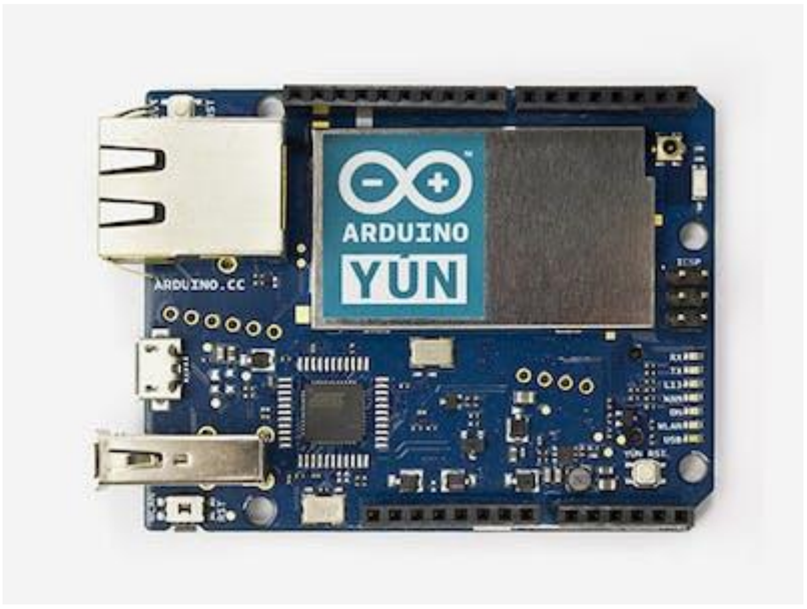


Figure 5.1 Arduino Yun

A microcontroller board was also used as a part of the hardware setup for the project. Arduino Yun is an Arduino board that consists of many modules that resemble typical functionalities that a normal IoT device constitutes, like Wi-Fi module and some pins to connect to environmental sensors. In this context, Arduino Yun was used as an IoT sensor to mimic a typical IoT device with an internet connection. This microcontroller board was mainly responsible for being a targeted device for incoming network attacks, while packet sniffing is done at the same time to capture the network packets for network classification

5.2 Software Setup

The software setup for this project involved the relevant software programs and tools: PyCharm, Anaconda, Docker and Flask. PyCharm and Anaconda were responsible for the training of the ML model, whereas Docker and Flask were used for executing the trained ML model in the form of a saved model file and creating a Python web application for the real-time network traffic prediction, respectively.

PyCharm is an integrated development environment (IDE) specifically built by JetBrains for Python programming language. It is required as a development platform for the process of ML model training, as the source codes for GAN were written in Python. PyCharm was chosen as the IDE in this project due to a built-in feature where a Python interpreter with the necessary Python libraries and modules can be selected to run for a specific use case. Having a specific interpreter creates separation of concern regarding any compatibility issues in certain Python libraries, as well as to isolate installed software packages and modules from other available Python interpreters.

Anaconda is an open-source distribution program for Python and R programming languages. Anaconda is mainly used for managing software libraries and packages that are dedicated to scientific computing fields like data science, machine learning and deep learning. With Anaconda installed, Python packages and libraries can be easily installed, upgraded, and removed, as well as new virtual environments can be created where packages and libraries can be used and managed separately from the host environment.

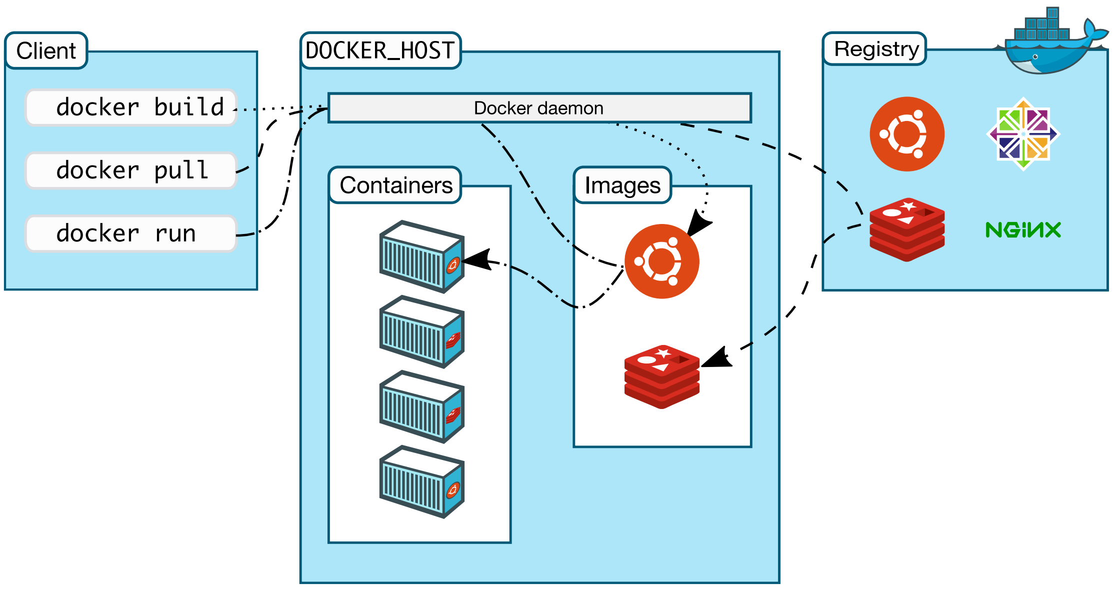


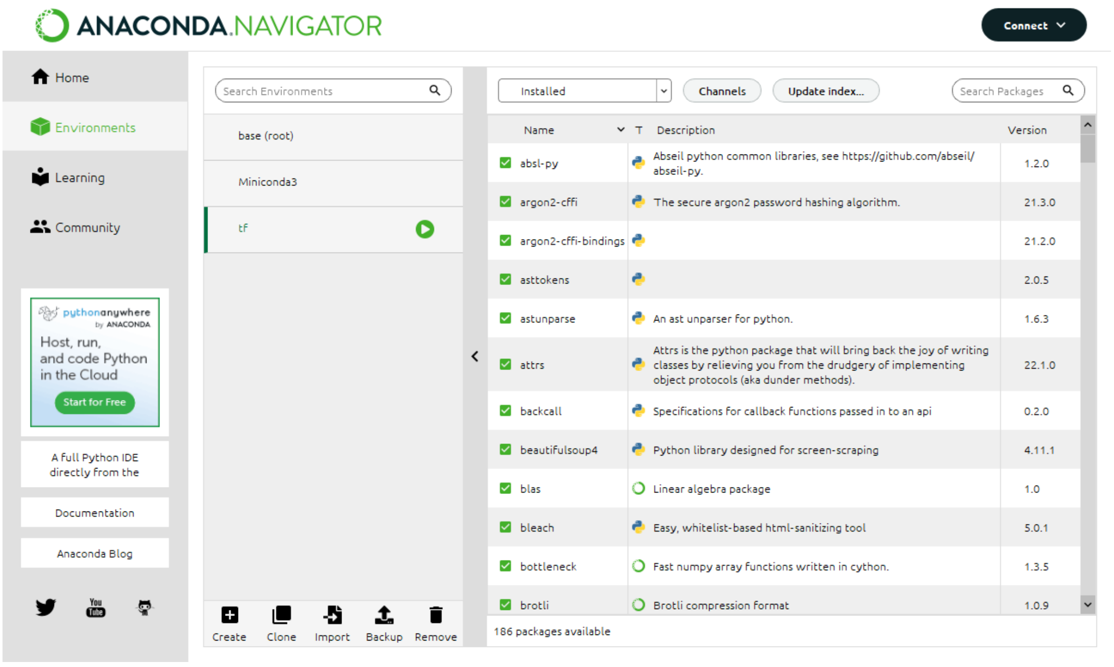
Figure 5.2 Docker architecture from *Docker*

Docker is a platform-as-a-service (PaaS) platform that utilises virtualisation in developing and running applications in separate packages called containers. The main advantage of using Docker is the separation of concern where a new infrastructure with a set of applications installed in it and is isolated from the main client infrastructure. As Docker uses a client-server architecture, the host machine that executes Docker commands, also known as Docker client, communicates with the Docker daemon, which is responsible for building and running Docker containers. The Docker client then can have the option of running and hosting a Docker container on the same system locally or connecting a Docker client to a remote Docker daemon via REST API and network interface. Docker is generally more lightweight and faster compared to hypervisor based VMs like VMWare and VirtualBox due to Docker containers sharing the same host OS rather than having a guest OS on top of the host OS. Therefore, much lesser computing resources are needed to run Docker containers compared to hosting hypervisor based VMs when it comes to system portability and managing application and services separately.

Flask, as a microframework for Python web development, simplifies the process of setting up a basic web application. In the case of getting the ML model prediction work as intended, Flask does the job as a simple web application is sufficient to simulate the IDS in predicting the network traffic patterns.

5.3 Setting and Configuration

There were numerous configurations and settings to be performed throughout the simulation process regarding the real-time IDS, from setting up environment variables for Python IDE in host machine to getting a web application up and running.

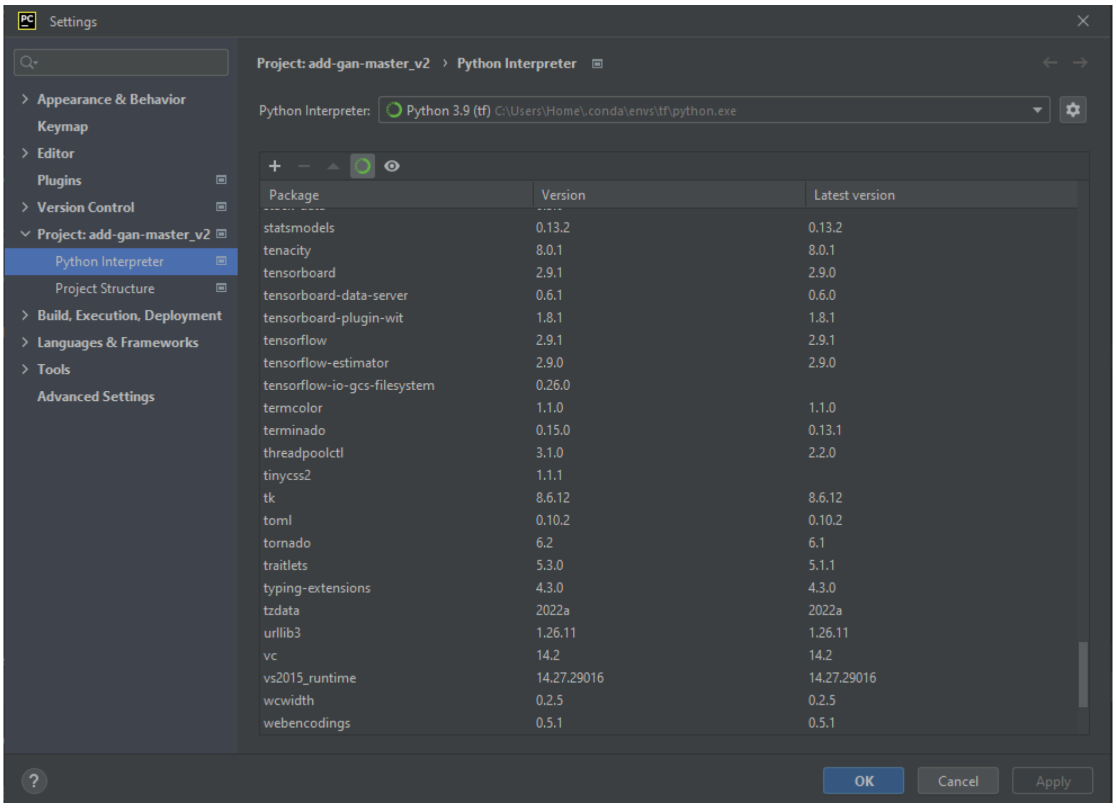


Name	T	Description	Version
✓ absl-py		Abseil python common libraries, see https://github.com/abseil/abseil-py.	1.2.0
✓ argon2-cffi		The secure argon2 password hashing algorithm.	21.3.0
✓ argon2-cffi-bindings			21.2.0
✓ asttokens			2.0.5
✓ astunparse		An ast unparser for python.	1.6.3
✓ attrs		Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	22.1.0
✓ backcall		Specifications for callback functions passed in to an api	0.2.0
✓ beautifulsoup4		Python library designed for screen-scraping	4.11.1
✓ blas		Linear algebra package	1.0
✓ bleach		Easy, whitelist-based html-sanitizing tool	5.0.1
✓ bottleneck		Fast numpy array functions written in cython.	1.3.5
✓ brotli		Brotli compression format	1.0.9

186 packages available

Figure 5.3 Anaconda in GUI version

The starting point of the configuration part for this project was to set up a Python interpreter and create a new Python virtual environment in the Windows host machine. Upon installing Anaconda program, the latest Python version will also be included as a part of the installation process. All virtual environments in the host system can be managed at the Environments section in the Anaconda Navigator, which is the GUI version of Anaconda. A new virtual environment was created, followed by the installation of the relevant Python packages and libraries like TensorFlow and PyPlot.



Project: add-gan-master_v2 › Python Interpreter

Python Interpreter: Python 3.9 (tf) C:\Users\Home\.conda\envs\tf\python.exe

Package	Version	Latest version
statsmodels	0.13.2	0.13.2
tenacity	8.0.1	8.0.1
tensorboard	2.9.1	2.9.0
tensorboard-data-server	0.6.1	0.6.0
tensorboard-plugin-wit	1.8.1	1.8.1
tensorflow	2.9.1	2.9.1
tensorflow-estimator	2.9.0	2.9.0
tensorflow-io-gcs-filesystem	0.26.0	
termcolor	1.1.0	1.1.0
terminado	0.15.0	0.13.1
threadpoolctl	3.1.0	2.2.0
tinycss2	1.1.1	
tk	8.6.12	8.6.12
toml	0.10.2	0.10.2
tornado	6.2	6.1
traitlets	5.3.0	5.1.1
typing-extensions	4.3.0	4.3.0
tzdata	2022a	2022a
urllib3	1.26.11	1.26.11
vc	14.2	14.2
vs2015_runtime	14.27.29016	14.27.29016
wcwidth	0.2.5	0.2.5
webencodings	0.5.1	0.5.1

Figure 5.4 Python Interpreter Settings in PyCharm IDE

The next configuration step would be setting up the environment variable and selecting the created virtual environment with the installed Python interpreter version for PyCharm IDE. Upon creating a new project in PyCharm, navigate to the existing interpreter section and select the virtual environment that was created previously. This would allow the IDE to run Python scripts and programs using the specified interpreter in the virtual environment. From there, all packages and modules installed in the particular virtual environment would also take effect within the virtual environment itself. In other words, this would separate the virtual environment from other environments such that the packages and libraries of different versions could be installed in the chosen virtual environment, not affecting the ones installed in the main environment.

The following step after the IDE configuration would be to convert the GAN source codes into an older version, so that the ML model could be executed for training without any runtime errors. Since the obtained source codes were written with TensorFlow 1.x version libraries, the current TensorFlow 2.x version (TensorFlow 2.9 as of May 2022) included many drastic changes and improvements from TensorFlow 1.x versions. Therefore, TensorFlow scripts [19] could be used to automatically convert all codes with TensorFlow libraries into a compatible-ready TensorFlow version, by adding *compat.v1* module in between all TensorFlow functions.

```
# install pip package manager
$ sudo apt-get install python-pip

# install virtualenv library module with pip installer
$ pip install virtualenv

# create a virtualenv for python 2 version
$ virtualenv -p /usr/bin/python2.7 virtualenv_name

# activate virtualenv
$ source virtualenv_name/bin/activate
```

Figure 5.5 Linux commands for creating virtual environment

As for the configurations regarding Docker and Flask, these would be performed in the Linux VM. Before setting up Docker and Flask, a Python virtual environment had to be created in the Linux VM. Having a separate virtual environment is important because the ML model deployment requires Python 2 versions to be able to run a Python client script that is responsible for communicating with the TensorFlow Serving model server from the Docker image.

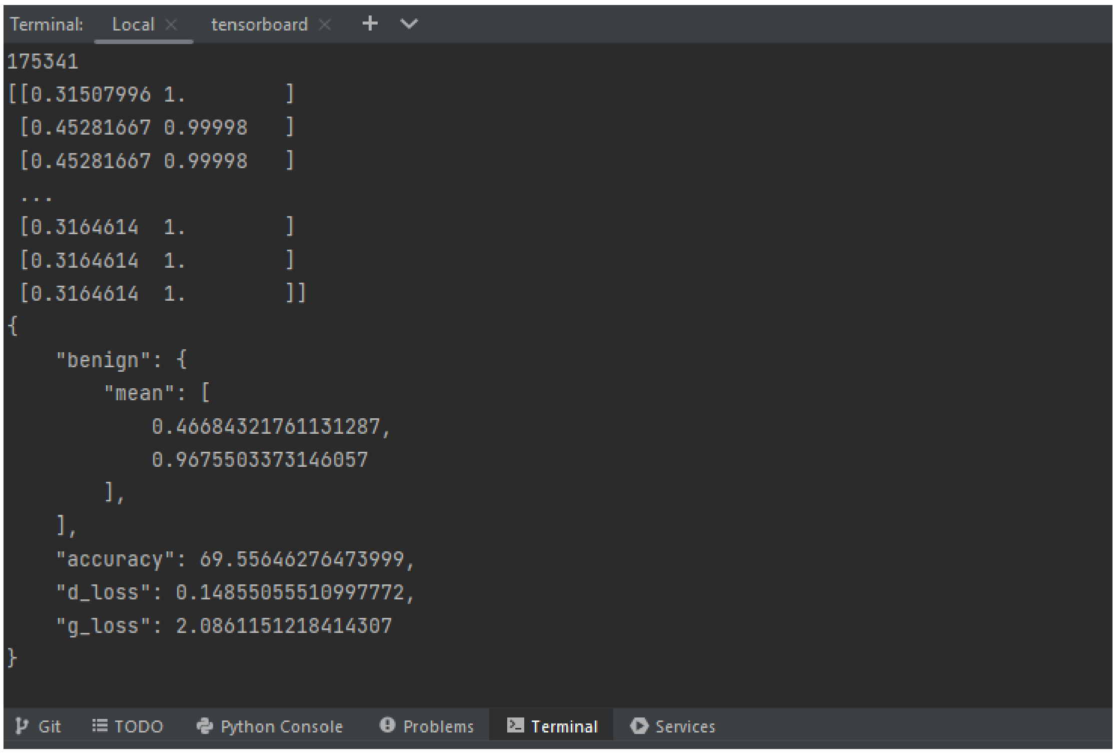
```
# install Docker library
$ pip install docker-io

# download TensorFlow Serving Docker image
$ docker pull tensorflow/serving

# install Flask library for web application
$ pip install Flask

# install gPRC predict client for TensorFlow Serving
$ pip install git+https://github.com/epigramai/tfserving-
python-predict-client.git
```

Figure 5.6 Linux commands for installing Docker and Flask libraries

To setup Docker in the Linux VM, Docker Engine, also known as docker-io for the package name, has to be installed in order to manage Docker containers and images. After that, the Docker image with TensorFlow Serving compiled on it is retrieved by downloading it from the official TensorFlow Docker repository hub. The downloaded Docker image would be used to start up a model server for TensorFlow Serving. Flask library should be installed using *pip* package manager tool to create a HTTP server for sending HTTP requests and response with the predict_client package designed specifically to be used with a model served by TensorFlow Serving. The predict_client package is needed to allow the ML model to perform prediction on the network traffic, as the TensorFlow Serving model server runs a gRPC service, which is a type of remote procedure call framework that simplifies client-server communications. With predict_client package installed, it helps to facilitate the communication as a middleman between the model server and the client application in sending HTTP and gRPC requests accordingly.

5.4 System Operation

First, the ML model had to be trained to be able to predict whether the network traffic is normal or malicious. The source codes pertaining to GAN were run in PyCharm IDE, with suitable hyperparameter values as the main parameters for yielding the best performance for the trained model possible. The main Python script to train the ML model was run to obtain the checkpoint files, which would be used for generating a saved model file to serve the model into deployment, and the performance metrics that determine the performance of the trained model.



```
175341
[[0.31507996 1.        ]
 [0.45281667 0.99998   ]
 [0.45281667 0.99998   ]
 ...
 [0.3164614  1.        ]
 [0.3164614  1.        ]
 [0.3164614  1.        ]]
{
    "benign": {
        "mean": [
            0.46684321761131287,
            0.9675503373146057
        ],
    ],
    "accuracy": 69.55646276473999,
    "d_loss": 0.1485505551097772,
    "g_loss": 2.0861151218414307
}
```

Figure 5.7 Output of trained model showing performance metrics

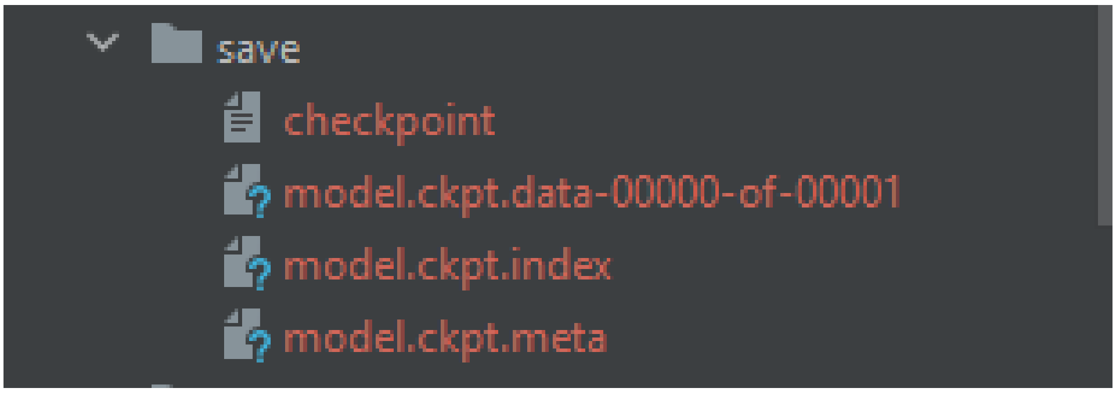
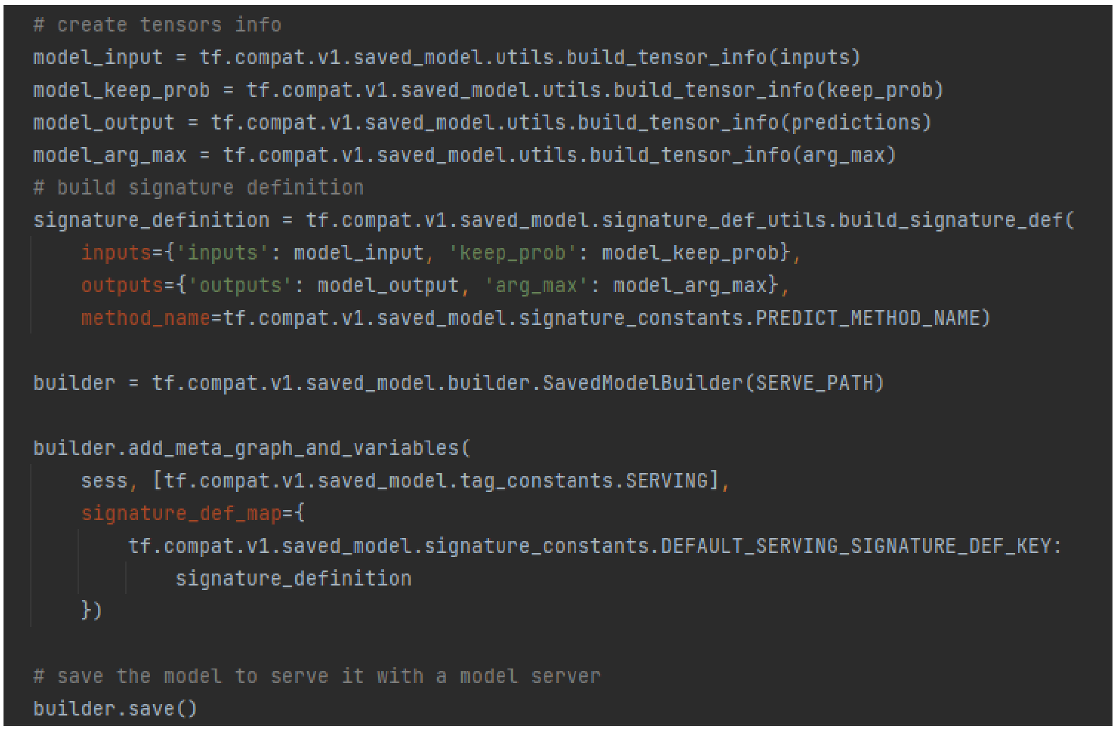


Figure 5.8 Generated model checkpoint files

The following step would be to create a saved model file from the checkpoint file previously. A Python script would be run to build a servable model from all the information and variables in the saved model checkpoint file. The resulting files generated from the script contain the important information regarding the model, like the calculated weight and bias values for each feature, and the algorithms needed to generate an outcome of the prediction.



```python
# create tensors info
model_input = tf.compat.v1.saved_model.utils.build_tensor_info(inputs)
model_keep_prob = tf.compat.v1.saved_model.utils.build_tensor_info(keep_prob)
model_output = tf.compat.v1.saved_model.utils.build_tensor_info(predictions)
model_arg_max = tf.compat.v1.saved_model.utils.build_tensor_info(arg_max)
# build signature definition
signature_definition = tf.compat.v1.saved_model.signature_def_utils.build_signature_def(
    inputs={'inputs': model_input, 'keep_prob': model_keep_prob},
    outputs={'outputs': model_output, 'arg_max': model_arg_max},
    method_name=tf.compat.v1.saved_model.signature_constants.PREDICT_METHOD_NAME)

builder = tf.compat.v1.saved_model.builder.SavedModelBuilder(SERVE_PATH)

builder.add_meta_graph_and_variables(
    sess, [tf.compat.v1.saved_model.tag_constants.SERVING],
    signature_def_map={
        tf.compat.v1.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            signature_definition
    })

# save the model to serve it with a model server
builder.save()
```

Figure 5.9 Code snippet for creating a saved model file for TensorFlow Serving

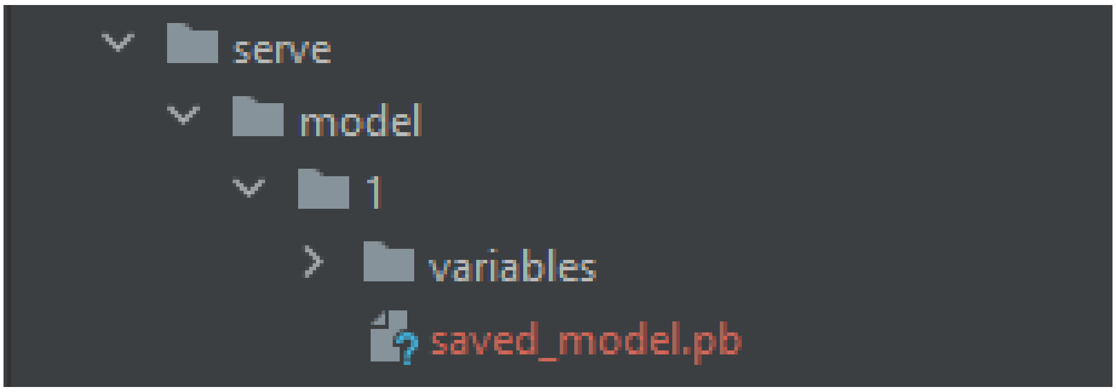
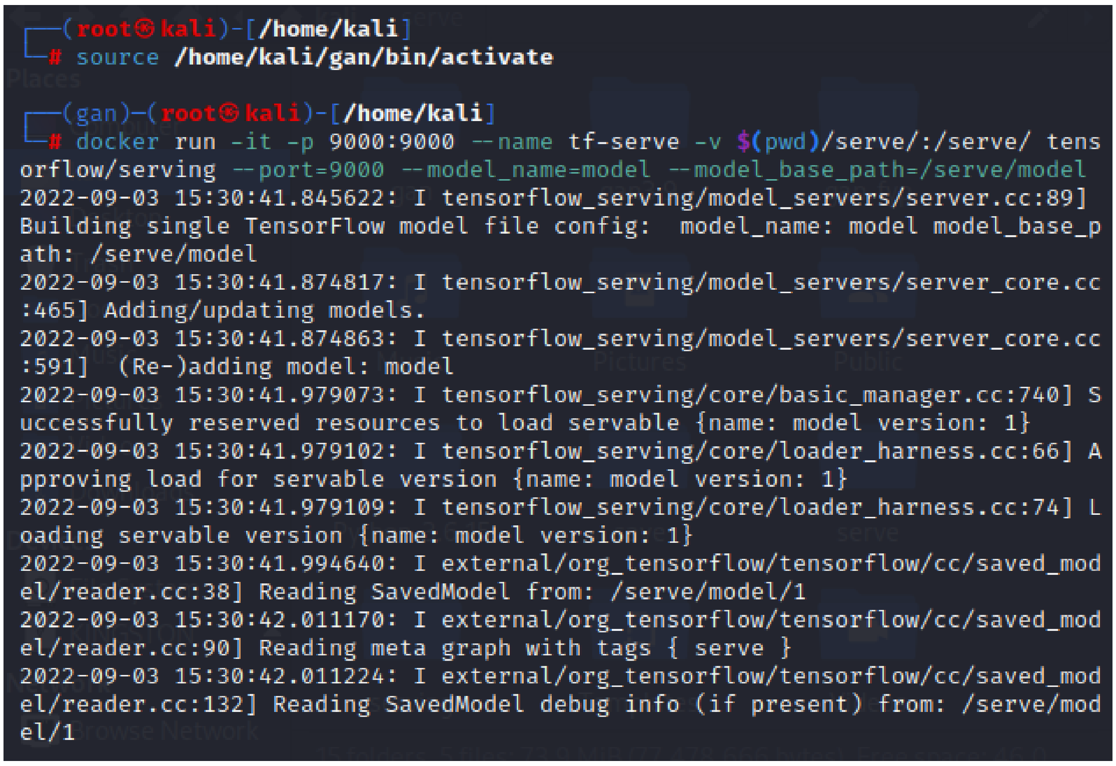


Figure 5.10 Saved model builder file

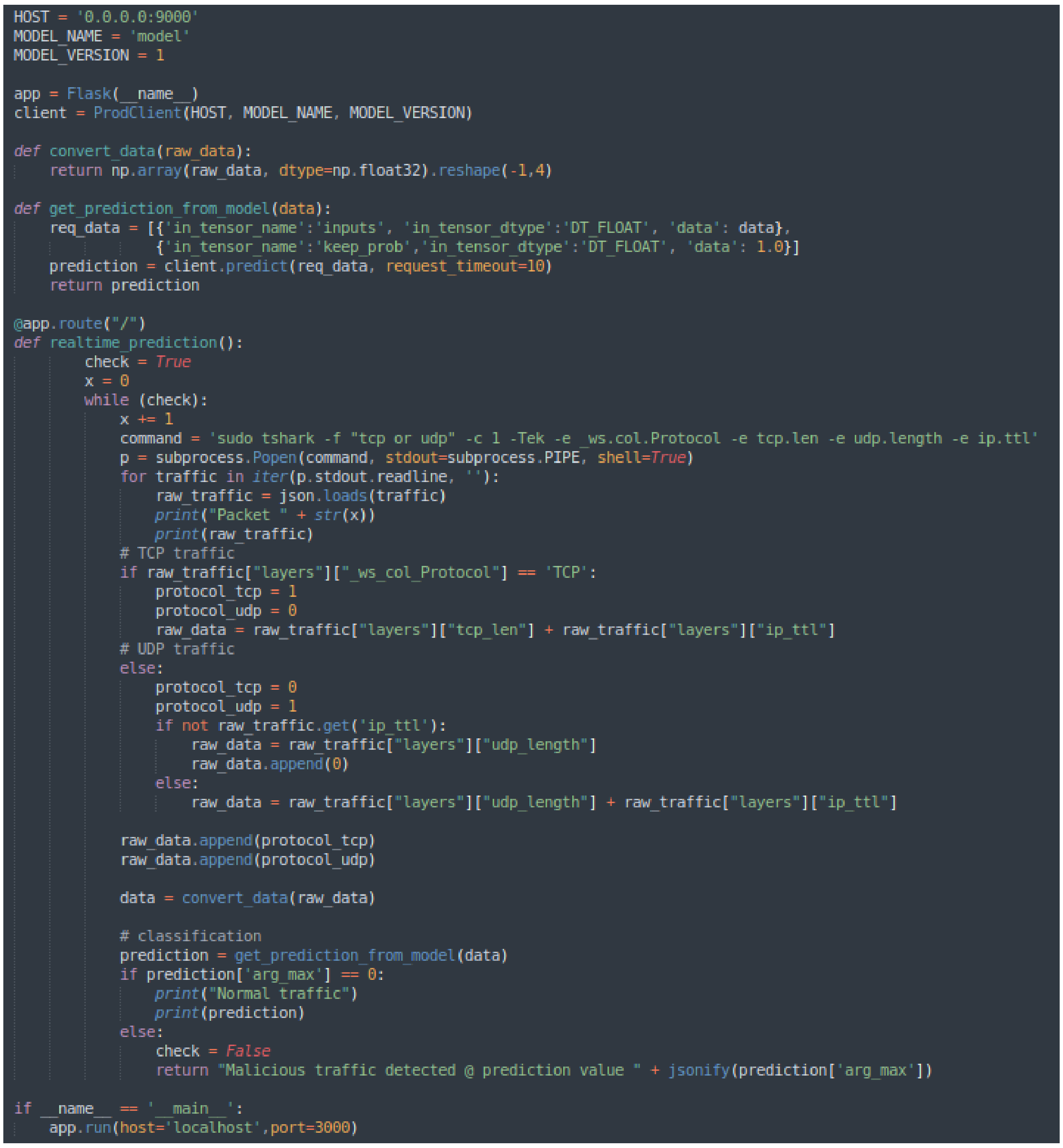
After generating the model builder files, a Python virtual environment was created in the Linux VM. The configurations for the creation of a new virtual environment could be referred to the one in Figure 5.5, followed by Figure 5.6 for installing all other necessary software packages and libraries prior to starting a Docker container. Upon doing so, a new Docker container was created with the parameters listed in Figure 5.10, and the downloaded TensorFlow Serving image was run inside the Docker container. As a result, the Docker container with the image would start to run and will be ready to be used for serving the ML model.



```
┌──(root㉿kali)-[/home/kali]
└─# source /home/kali/gan/bin/activate

┌──(gan)─(root㉿kali)-[/home/kali]
└─# docker run -it -p 9000:9000 --name tf-serve -v $(pwd)/serve/:/serve/ tens
orflow/serving --port=9000 --model_name=model --model_base_path=/serve/model
2022-09-03 15:30:41.845622: I tensorflow_serving/model_servers/server.cc:89]
Building single TensorFlow model file config:  model_name: model model_base_p
ath: /serve/model
2022-09-03 15:30:41.874817: I tensorflow_serving/model_servers/server_core.cc
:465] Adding/updating models.
2022-09-03 15:30:41.874863: I tensorflow_serving/model_servers/server_core.cc
:591]  (Re-)adding model: model
2022-09-03 15:30:41.979073: I tensorflow_serving/core/basic_manager.cc:740] S
uccessfully reserved resources to load servable {name: model version: 1}
2022-09-03 15:30:41.979102: I tensorflow_serving/core/loader_harness.cc:66] A
pproving load for servable version {name: model version: 1}
2022-09-03 15:30:41.979109: I tensorflow_serving/core/loader_harness.cc:74] L
oading servable version {name: model version: 1}
2022-09-03 15:30:41.994640: I external/org_tensorflow/tensorflow/cc/saved_mod
el/reader.cc:38] Reading SavedModel from: /serve/model/1
2022-09-03 15:30:42.011170: I external/org_tensorflow/tensorflow/cc/saved_mod
el/reader.cc:90] Reading meta graph with tags { serve }
2022-09-03 15:30:42.011224: I external/org_tensorflow/tensorflow/cc/saved_mod
el/reader.cc:132] Reading SavedModel debug info (if present) from: /serve/mod
el/1
```

Figure 5.11 Running TensorFlow Serving image in Docker container

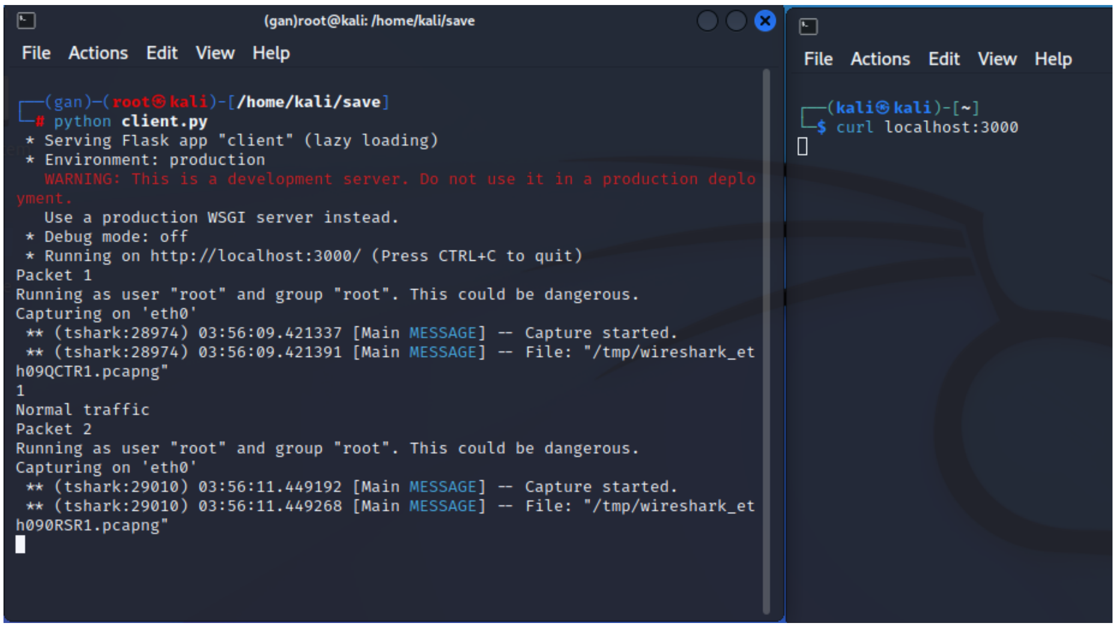
Once the Docker container started running, another Python script that is responsible for the real-time classification of network traffic will be executed in the Linux VM. The script included the necessary codes for starting up a simple Flask web application, so that the prediction of network traffic based on the captured network packets will occur. It would also perform packet sniffing on the network with Tshark command, pre-process the captured network packets to a NumPy array format, and feed the data into the client prediction library that was responsible for sending requests to the model server for prediction results.



```python
HOST = '0.0.0.0:9000'
MODEL_NAME = 'model'
MODEL_VERSION = 1

app = Flask(__name__)
client = ProdClient(HOST, MODEL_NAME, MODEL_VERSION)

def convert_data(raw_data):
    return np.array(raw_data, dtype=np.float32).reshape(-1,4)

def get_prediction_from_model(data):
    req_data = [{'in_tensor_name':'inputs', 'in_tensor_dtype':'DT_FLOAT', 'data': data},
                {'in_tensor_name':'keep_prob','in_tensor_dtype':'DT_FLOAT', 'data': 1.0}]
    prediction = client.predict(req_data, request_timeout=10)
    return prediction

@app.route("/")
def realtime_prediction():
        check = True
        x = 0
        while (check):
            x += 1
            command = 'sudo tshark -f "tcp or udp" -c 1 -Tek -e _ws.col.Protocol -e tcp.len -e udp.length -e ip.ttl'
            p = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
            for traffic in iter(p.stdout.readline, ''):
                raw_traffic = json.loads(traffic)
                print("Packet " + str(x))
                print(raw_traffic)
            # TCP traffic
            if raw_traffic["layers"]["_ws_col_Protocol"] == 'TCP':
                protocol_tcp = 1
                protocol_udp = 0
                raw_data = raw_traffic["layers"]["tcp_len"] + raw_traffic["layers"]["ip_ttl"]
            # UDP traffic
            else:
                protocol_tcp = 0
                protocol_udp = 1
                if not raw_traffic.get('ip_ttl'):
                    raw_data = raw_traffic["layers"]["udp_length"]
                    raw_data.append(0)
                else:
                    raw_data = raw_traffic["layers"]["udp_length"] + raw_traffic["layers"]["ip_ttl"]

            raw_data.append(protocol_tcp)
            raw_data.append(protocol_udp)

            data = convert_data(raw_data)

            # classification
            prediction = get_prediction_from_model(data)
            if prediction['arg_max'] == 0:
                print("Normal traffic")
                print(prediction)
            else:
                check = False
                return "Malicious traffic detected @ prediction value " + jsonify(prediction['arg_max'])

if __name__ == '__main__':
    app.run(host='localhost',port=3000)
```

Figure 5.12 Python code for real-time prediction with Flask

Upon executing the client prediction script, the Flask web application would start running. Whenever the URL of the web application was visited, the Tshark packet sniffing command would start executing along with the client prediction codes to display the classification outcome of each captured network packet. The output for the classification of each network packet can be ‘Normal traffic’ or ‘Malicious traffic’ depending on the prediction results from the model server.


(gan)root@kali: /home/kali/save
File Actions Edit View Help
File Actions Edit View Help

```
┌──(gan)─(root㉿kali)-[/home/kali/save]
└─# python client.py
 * Serving Flask app "client" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deplo
yment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://localhost:3000/ (Press CTRL+C to quit)
Packet 1
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
 ** (tshark:28974) 03:56:09.421337 [Main MESSAGE] -- Capture started.
 ** (tshark:28974) 03:56:09.421391 [Main MESSAGE] -- File: "/tmp/wireshark_et
h09QCTR1.pcapng"
1
Normal traffic
Packet 2
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
 ** (tshark:29010) 03:56:11.449192 [Main MESSAGE] -- Capture started.
 ** (tshark:29010) 03:56:11.449268 [Main MESSAGE] -- File: "/tmp/wireshark_et
h090RSR1.pcapng"
```

```
┌──(kali㉿kali)-[~]
└─$ curl localhost:3000
```

Figure 5.13 Network traffic prediction with Flask

5.5 Implementation Issues and Challenges

There were a few challenges as to implementing the relevant setup on the IDS. One of which was the data pre-processing of the UNSW-NB15 dataset that is important prior to feeding the data into the ML model for training and testing. Most of the derived features in the dataset had very strong correlation with the given labels that indicate whether the network traffic is benign or malicious. Also, many features like source and destination IP and ports were dropped due to the redundancy of the data and the irrelevancy in determining the state of the network traffic. The remaining features were further reduced due to the packet sniffing method used during which the live capture functionality has very limited capabilities in extracting the necessary features of network packets for IDS prediction.

The setup of TensorFlow Serving model server was also a big challenge for this real-time IDS implementation. As there are other methods as to deploy a ML model more efficiently and compatibly, the source codes for GAN were specifically written with an older version of TensorFlow library. This would result in limited options for deploying the said model, and more configuration steps were required to set up a functioning model server for model prediction compared to codes written with TensorFlow 2.x version libraries, which provide more APIs to simplify the deployment of an ML model.

As for the packet sniffing method used, Tshark does not offer much flexibility when it comes to capturing live packets in a network interface. Since the UNSW-NB15 dataset had most features derived from the algorithms written in tools like Argus and Bro IDS, all of which were not publicly accessible, network packet analyser tools like Tshark could only extract the most basic features from the captured network packets. Although the display capture feature in Tshark consists of more comprehensive options that can decode captured network packets in a more detailed manner, it does not serve the purpose of simulating the nature of what a real-time IDS should be. Therefore, the live capture feature of Tshark was used instead.

5.6 Concluding Remark

This chapter concludes the whole simulation process of the IDS, as well as covers all the necessary information, from the hardware and software setup to the system operation of the IDS in details.

Chapter 6
System Evaluation and Discussion

To determine the performance and the effectiveness of the real-time IDS, system evaluations have to be performed. This chapter will include the results obtained from the testing setup and some performance metrics for the approach, as well as the project challenges encountered, and the evaluation of the objectives mentioned in the report.

6.1 System Testing and Performance Metrics

Regarding the testing procedure of a typical IDS, the most common performance metric that is used to determine the effectiveness of a trained model in model prediction is the confusion matrix. Confusion matrix is a table measurement for ML classification problem where the output can be two or more classes, depending on the trained model. It is normally shown with at least 4 different combinations of two types of values: actual and predicted. The confusion matrix can be classified as true labels of normal and attack categories, and predicted labels of normal and attack categories, with 0 indicating normal and 1 indicating malicious. The predicted values in the confusion matrix are true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN).

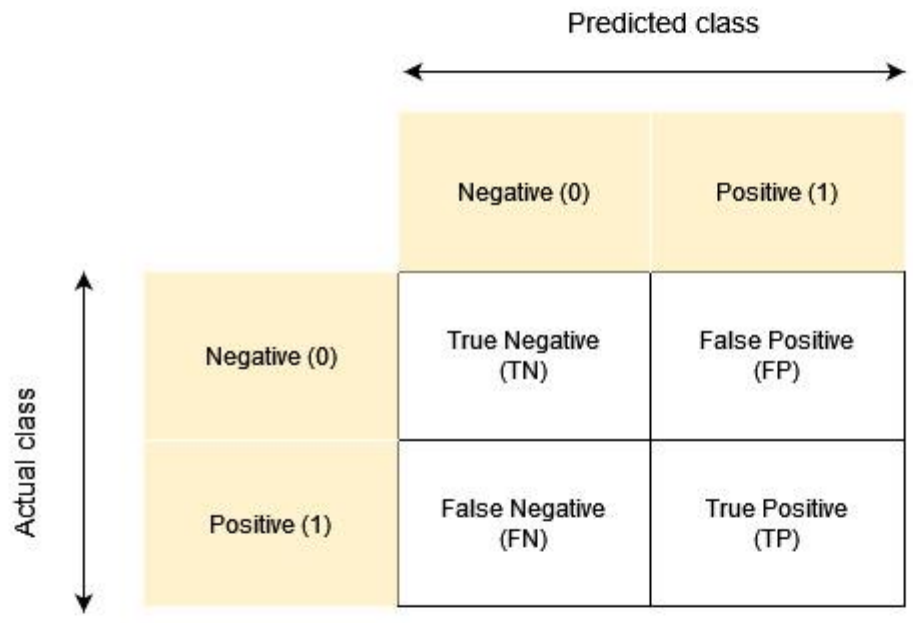


Figure 6.1 Confusion matrix

- TP (predicted malicious traffic correctly): predicts 1 and the actual label is 1.
- FP (predicted malicious traffic incorrectly): predicts 1 and the actual label is 0.
- TN (predicted normal traffic correctly): predicts 0 and the actual label is 0.
- FN (predicted normal traffic incorrectly): predicts 0 and the actual label is 1.

The confusion matrix values can be calculated through math to obtain four other metrics: recall, precision, accuracy, and F1-score. The equations for all four metrics are shown below:

$$\text{Accuracy} = \frac{TP+TN}{Total\ Samples}$$

$$\text{Recall (R)} = \frac{TP}{TP+FN}$$

$$\text{Precision (P)} = \frac{TP}{TP+FP}$$

$$\text{F1-Score} = 2 \times \frac{R \times P}{R+P}$$

Accuracy calculates the total number of correct predictions out of all predictions. Recall determines the number of correct predictions from all the positive classes, whereas precision calculates how many are positive out of the total number of predictions classified as positive, and F1-score is derived from the average value of recall and precision. The performance of a model is good when all values are as high as possible at an ideal rate, but there will always be some discrepancies between each of them and every model is different in some ways.

6.2 Testing Setup and Result

The performance of the GAN model was evaluated after averaging the number of training times with tuned main hyperparameters shown in Table 6.1. The model performance was also illustrated using the confusion matrix, accuracy, precision, recall, F1-score, discriminator loss and generator loss.

Table 6.1 Tuned hyperparameter values for GAN model

Hyperparameter	**Value**
Number of epochs	100
Batch size	1000
Discriminator learning rate	0.01
Generator learning rate	0.001

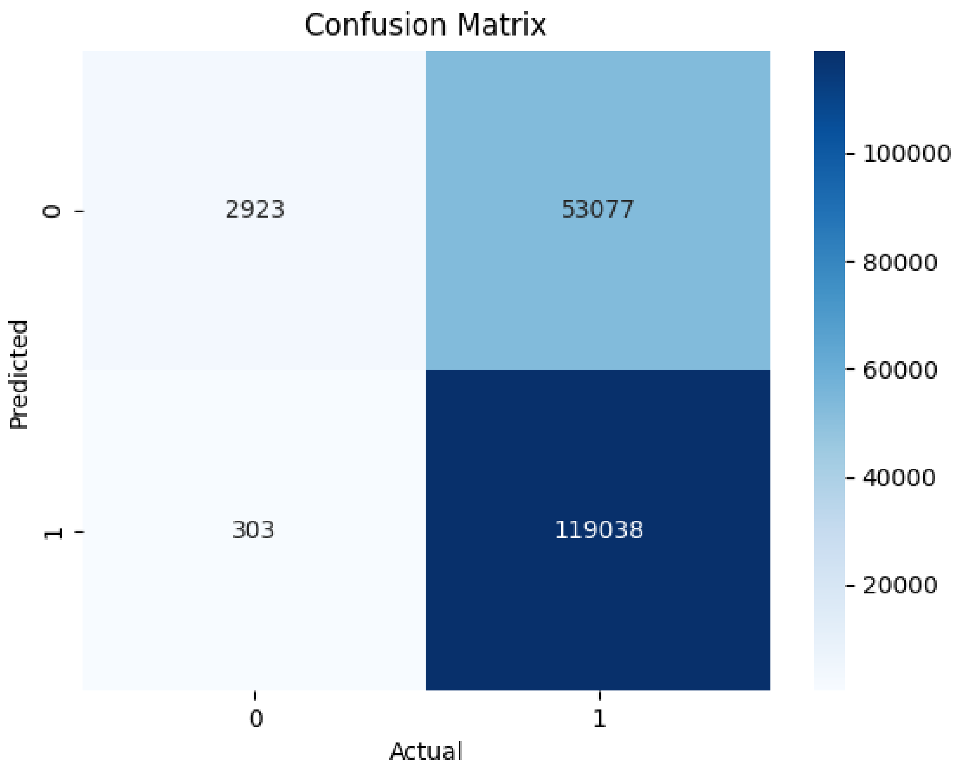


Figure 6.2 Confusion matrix of UNSW-NB15 testing dataset

Table 6.2 Testing set results

Metrics	Testing set values
Accuracy	0.6956
Recall	0.9975
Precision	0.6916
F1-score	0.8169

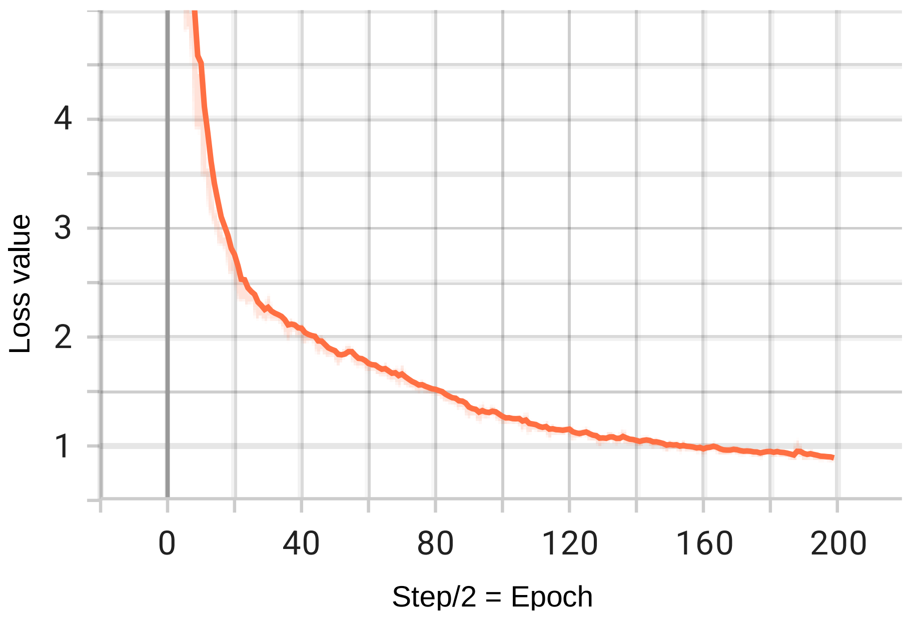


Figure 6.3 Discriminator loss

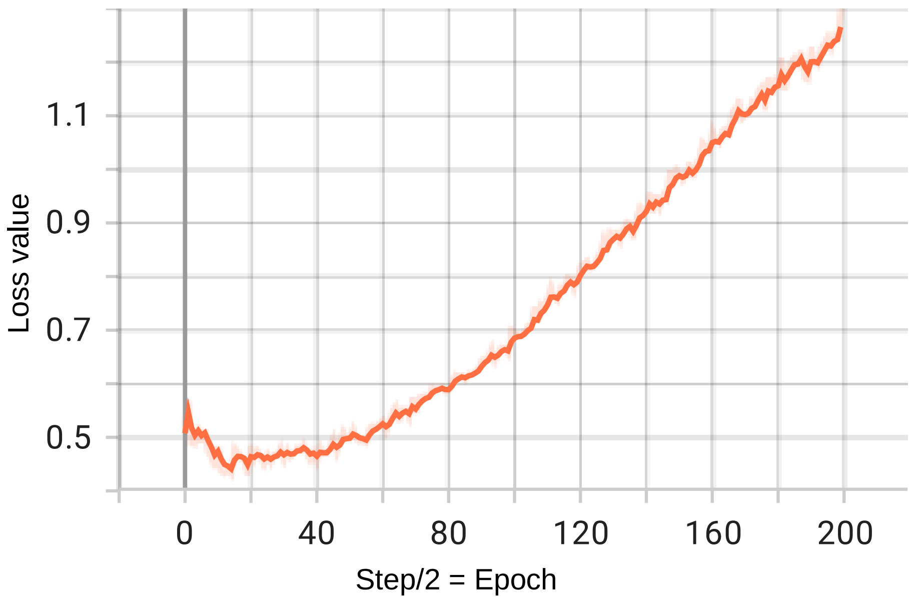
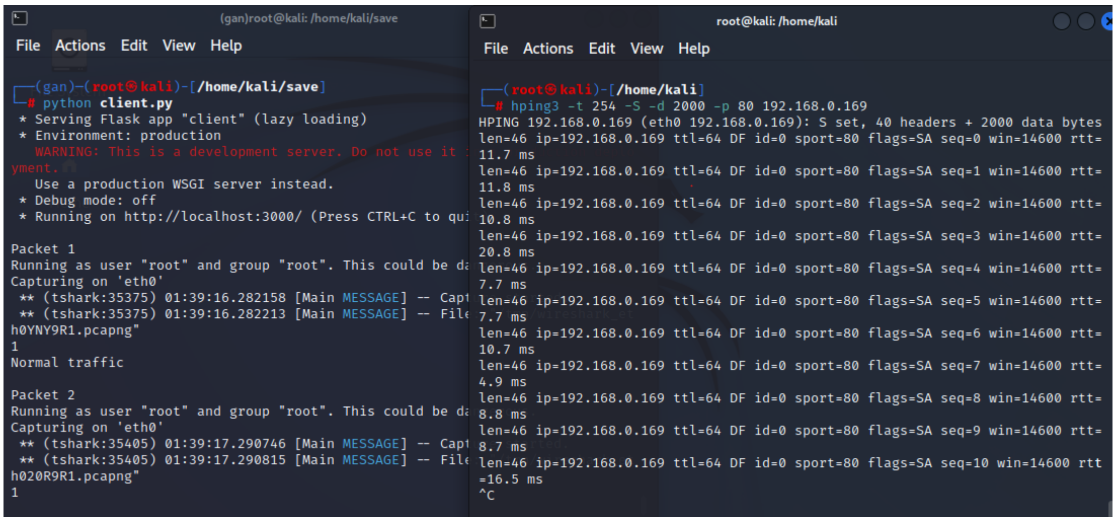


Figure 6.4 Generator loss

Based on the testing set results in Table 6.2, the accuracy of the trained model with tuned hyperparameters was around 69%, with many predictions skewing towards the FP region in the confusion matrix. Although an ideal confusion matrix requires most predictions to be aligned diagonally to the left, the testing set from the UNSW-NB15 testing dataset could only achieve around 70% accuracy at best with constant hyperparameter tuning.

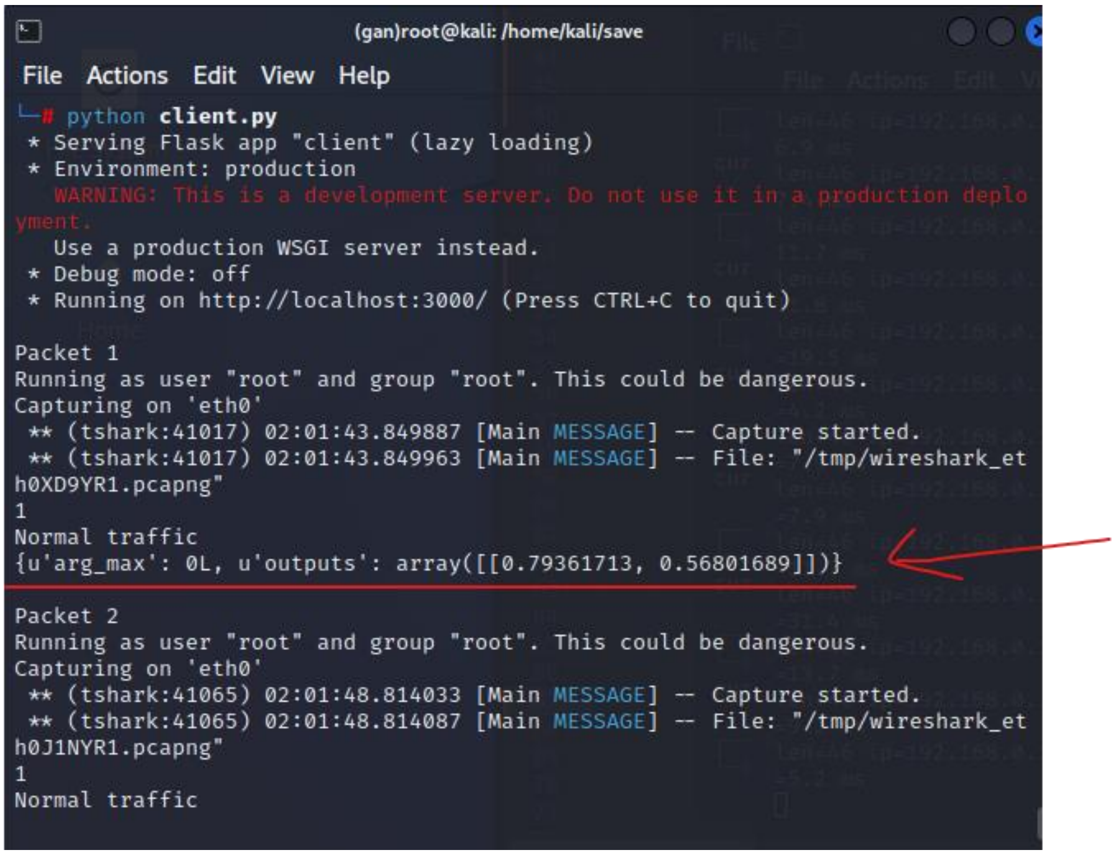
According to Figure 6.3 and Figure 6.4, the discriminator loss reduced steadily, whereas the generator loss increases steadily. In a GAN model, the ideal situation for both generator and discriminator losses are for them to converge and stabilise after a certain number of epochs. However, based on the figures, both losses will start to diverge after approximately 75 epochs (150 steps divided by 2). This might mean that the convergence for both sub-models is very difficult to achieve based on the hyperparameters. From another point of view, the training of the model can be considered to be completed, as the generator cannot be further improved anymore due to the discriminator not being able to be fooled by the generator. In other words, the discriminator gets better at distinguishing fake data from real input data. Nevertheless, for each time the model is trained, the values of both losses will vary, as the resulting weights and biases for each training epoch will also vary due to the randomly assigned weight and bias values at the start of model training.


(gan)root@kali: /home/kali/save
File Actions Edit View Help
root@kali: /home/kali
File Actions Edit View Help

```
┌──(gan)─(root㉿kali)-[/home/kali/save]
└─# python client.py
 * Serving Flask app "client" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it i
yment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://localhost:3000/ (Press CTRL+C to qui

Packet 1
Running as user "root" and group "root". This could be da
Capturing on 'eth0'
 ** (tshark:35375) 01:39:16.282158 [Main MESSAGE] -- Capt
 ** (tshark:35375) 01:39:16.282213 [Main MESSAGE] -- File
h0YNY9R1.pcapng"
1
Normal traffic

Packet 2
Running as user "root" and group "root". This could be da
Capturing on 'eth0'
 ** (tshark:35405) 01:39:17.290746 [Main MESSAGE] -- Capt
 ** (tshark:35405) 01:39:17.290815 [Main MESSAGE] -- File
h020R9R1.pcapng"
1
```

```
┌──(root㉿kali)-[/home/kali]
└─# hping3 -t 254 -S -d 2000 -p 80 192.168.0.169
HPING 192.168.0.169 (eth0 192.168.0.169): S set, 40 headers + 2000 data bytes
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=14600 rtt=
11.7 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=1 win=14600 rtt=
11.8 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=2 win=14600 rtt=
10.8 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=3 win=14600 rtt=
20.8 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=4 win=14600 rtt=
7.7 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=5 win=14600 rtt=
7.7 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=6 win=14600 rtt=
10.7 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=7 win=14600 rtt=
4.9 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=8 win=14600 rtt=
8.8 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=9 win=14600 rtt=
8.7 ms
len=46 ip=192.168.0.169 ttl=64 DF id=0 sport=80 flags=SA seq=10 win=14600 rtt
=16.5 ms
^C
```

Figure 6.5 TCP flood attack performed while classifying data packets



```
└─# python client.py
 * Serving Flask app "client" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deplo
yment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://localhost:3000/ (Press CTRL+C to quit)

Packet 1
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
 ** (tshark:41017) 02:01:43.849887 [Main MESSAGE] -- Capture started.
 ** (tshark:41017) 02:01:43.849963 [Main MESSAGE] -- File: "/tmp/wireshark_et
h0XD9YR1.pcapng"
1
Normal traffic
{u'arg_max': 0L, u'outputs': array([[0.79361713, 0.56801689]])}

Packet 2
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
 ** (tshark:41065) 02:01:48.814033 [Main MESSAGE] -- Capture started.
 ** (tshark:41065) 02:01:48.814087 [Main MESSAGE] -- File: "/tmp/wireshark_et
h0J1NYR1.pcapng"
1
Normal traffic
```

Figure 6.6 Classification results

As for the real-time prediction process of the IDS, the outcome of the prediction based on the underlined part in Figure 6.6 showed the ‘outputs’ key of the JSON returned an array of two probability values. To classify the data packet, the larger value from the two elements in the array will be taken and the result will be displayed in ‘arg_max’ key of the JSON output. Based on the result in Figure 6.6, the first element of the ‘outputs’ array was larger than the second element, and so the value of 0 was returned for the ‘arg_max’ JSON key, indicating a normal traffic. If the second element of the ‘outputs’ array was larger compared to the first one, value of ‘arg_max’ would be 1 and classified as a malicious traffic. However, it was not able to detect any signs of possible network intrusions while network attacks like DDoS and port scan were performed using hping3 and Nmap respectively.

Since the prediction of the network traffic depends on the trained weight and bias values of all the neurons in all the input, hidden and output layers, the output of the prediction will be constant based on the same given input data from the captured packets. With the Tshark command having the option of “-c 1” included, it is only able to capture a network packet to allow the captured data to be pre-processed for the classification function to work. Although the packet sniffing process occurs infinitely when the URL of the Flask web application is visited, packet sniffing has to be done with 1 packet capture at a time for data pre-processing, hence some data packets were not captured in the process.

6.3 Project Challenges

One of the main challenges in this project is the training of deep learning framework models like GAN. Generative models like GAN are typically very difficult to train as there is no balance for when the discriminator and generator networks achieve convergence and no clear indicators as to when the model is considered to be finished training. In that case, the occurrence of overfitting or underfitting of the model during the training process will be hard to identify as well. Aside from that, the hyperparameter tuning process was time-consuming, as it is a trial-and-error method where constant value adjustments for each of the hyperparameters are needed in hopes for a stable, trained model. There are many factors to be considered when it comes to training a stable GAN model, and each of them will have a direct impact on the performance of the GAN model, and all models that implement some sort of deep learning framework.

Another challenge faced throughout this project is the real-time aspect of the network classification. The packet sniffing method used was quite limited in terms of filtering very specific packet information compared to the display filter option, where it is done using a saved capture file. In addition, the number of features able to be extracted with Tshark live capture method was very little, as most features in the UNSW-NB15 dataset are complex features. Hence, all the complex features had to be removed to accommodate to the basic features only for packet sniffing to work as intended. With all things put into place, the classification outcomes from the real-time network traffic prediction of the IDS were not desirable when it was used as in a simulation scenario.

6.4 Objectives Evaluation

The objectives in the project were not able to be realised to a certain extent, as the real-time IDS was not able to detect network intrusions most of the times. Although the GAN model could be trained successfully and the implementation of the real-time IDS was able to be conducted, the end-results obtained were not as desirable as to be expected for an effective IDS in a deployment environment.

6.5 Concluding Remark

This chapter concludes the testing of the real-time IDS and the simulation results from the IDS prediction, along with the metrics used in evaluating the performance of the trained GAN model.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

This research explores the implementation of real-time IDS with a deep learning-based ML framework. The importance of a security technology system like IDS in the IoT domain, with medical devices included, must be considered to further improve the security aspects of the IoT devices. With the real-time IDS implementation, it is shown to be able to capture network data and identify any network intrusions, while being hosted in a web application to provide access for everyone in the medical field, corporate or even personal usage. The implementation procedures and techniques used in the project could be replicated and modified, if necessary, to further enhance the performance and real-time aspect of the IDS. The use of a real-time IDS is suitable for anyone with less knowledge on the technical aspects of the implementation and can be deployed on any equipment or machines with relatively low-power consumption. A real-time IDS would be very useful in scenarios when anyone needs to monitor the network traffic and identify for possible intrusions in a relatively simple manner. This project is done to stress the importance of a real-time IDS in adding an extra layer of security for IoT devices, as well as to gear the development and further improvements towards the implementation of real-time IDS in real-world scenarios.

7.2. Recommendation

The implementation of the real-time IDS in this project could be further improved in several aspects. One of which is the use of other various ML models or those with deep learning frameworks to potentially improve the performance of the trained model. Other than that, the use of other publicly datasets like NSL-KDD and KDD99 datasets could be used to evaluate the performance of the trained model, as well as other publicly available and reliable network intrusion datasets in the future. The feature extraction aspect in the packet sniffing procedure could also be improved by adding more relevant features for IDS testing, provided that the selected dataset contains more features that are extractable using appropriate packet sniffing tools. In addition, the features in most public network intrusion datasets could be further clarified on the ways the features are exactly extracted to obtain those features, so that the testing of real-time IDS would be more effective in the future.

References

[1] Ordr, "10 internet of things (IoT) healthcare examples," [Online]. Available: https://ordr.net/article/iot-healthcare-examples/.

[2] J.-P. A. Yaacoub, M. Noura, H. N. Noura, O. Salman, E. Yaacoub, R. Couturier and A. Chehab, "Securing Internet of Medical Things Systems: Limitations, Issues and Recommendations," *Future Generation Computer Systems,* vol. 105, pp. 581-606, 2020.

[3] L. Pycroft and T. Z. Aziz, "Security of implantable medical devices with wireless connections: The dangers of cyber-attacks," *Expert Review of Medical Devices,* pp. 403-406, 2018.

[4] A. Khraisat and A. Alazab, "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges," *Cybersecurity,* 2021.

[5] A. Nahapetian, "Side-Channel Attacks on Mobile and Wearable Systems," in *13th IEEE Annual Consumer Communications and Networking Conference (CCNC)*, Las Vegas, 2016.

[6] C. Shepherd, K. Markantonakis, N. van Heijningen, D. Aboulkassimi, C. Gaine, T. Heckmann and D. Naccache, "Physical Fault Injection and Side-Channel Attacks on Mobile Devices: A Comprehensive Survey," *Computerse & Security,* 2021.

[7] E. Hodo, X. Bellekens, A. Hamilton, P.-L. Dubouilh, E. Iorkyase, C. Tachtatzis and R. Atkinson, "Threat analysis of IoT networks using artificial neural network intrusion detection system," in *International Symposium on Networks, Computers and Communications (ISNCC)*, Yasmine Hammamet, 2016.

[8] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto and K. Sakurai, "Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture," in *4th IEEE Cyber Science and Technology Congress*, Fukuoka, 2020.

[9] C. Cervantes, D. Poplade, M. Nogueira and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6LoWPAN for Internet of Things," in *IFIP/IEEE International Symposium on Integrated Network Management (IM),* Ottawa, 2015.

[10] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set)," in *Military Communications and Information Systems Conference (MilCIS)*, 2015.

[11] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," in *Information Security Journal: A Global Perspective*, 2016.

[12] N. Moustafa, J. Slay and G. Creech, "Novel Geometric Area Analysis Technique for Anomaly Detection Using Trapezoidal Area Estimation on Large-Scale Networks," in *IEEE Transactions on Big Data*, 2017.

[13] N. Moustafa, G. Creech and J. Slay, "Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models," in *Data Analytics and Decision Support for Cybersecurity*, Springer, Cham, 2017.

[14] M. Sarhan, S. Layeghy, N. Moustafa and M. Portmann, "NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems," in *Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020*, 2020.

[15] Tshark, "Capture Lifecycle with Tshark," [Online]. Available: https://tshark.dev/.

[16] TensorFlow, "Serving Models," [Online]. Available: https://www.tensorflow.org/tfx/guide/serving.

[17] Flask, "A Minimal Application," [Online]. Available: https://flask.palletsprojects.com/en/2.2.x/quickstart/.

[18] Z. Blasingame, "Github," [Online]. Available: https://github.com/zblasingame/ADD-GAN.

[19] TensorFlow, "Automatically rewrite TF 1.x and compat.v1 API symbols," [Online]. Available: https://www.tensorflow.org/guide/migrate/upgrade.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 4**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE

- Planned the structure flow of the implementation

2. WORK TO BE DONE

- Start the system implementation
- Continue the prototype development

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- Started with initial progression

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 6**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE

- Finished debugging source codes
- Done configurations for training the model

2. WORK TO BE DONE

- Research for a suitable network dataset
- Start setting up TensorFlow model server

3. PROBLEMS ENCOUNTERED

- Some runtime and compatibility issues on training ML source codes

4. SELF EVALUATION OF THE PROGRESS

- Able to keep up with current progress

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 8**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE
- Trained ML model
- Done initial setup on Linux VM

2. WORK TO BE DONE
- Perform hyperparameter tuning for the model
- Modify network dataset features
- Set up model server for serving the trained model

3. PROBLEMS ENCOUNTERED
- Configuration issues for model server setup
- Trained model returned abnormal results

4. SELF EVALUATION OF THE PROGRESS
- Able to cope with self-assigned tasks within the timeframe

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 10**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE

- Reduced network dataset features
- Researched on Tshark operations

2. WORK TO BE DONE

- Continuously perform hyperparameter tuning
- Research on Flask usage
- Start writing report

3. PROBLEMS ENCOUNTERED

- Encountered configurations errors in model server setup

4. SELF EVALUATION OF THE PROGRESS

- Felt a bit behind the planned schedule, but still manageable

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 12**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE
- Done model server setup with Docker
- Done Flask web app setup
- Completed most parts of the report

2. WORK TO BE DONE
- Deploy the IDS in Linux VM
- Test the IDS
- Finish the remaining parts of the report

3. PROBLEMS ENCOUNTERED
- IDS could not classify network traffic correctly

4. SELF EVALUATION OF THE PROGRESS
- Able to complete the implementation part

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	**Study week no.: 13**
Student Name & ID: Joshua Phang Jen Hoe, 18ACB06775	
Supervisor: Dr Gan Ming Lee	
Project Title: Real-Time Intrusion Detection System in IoT Medical Devices	

1. WORK DONE

- Finished writing the report
- Deployed IDS in VM

2. WORK TO BE DONE

- Finalise the report for submission

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- Able to finish up all self-assigned tasks

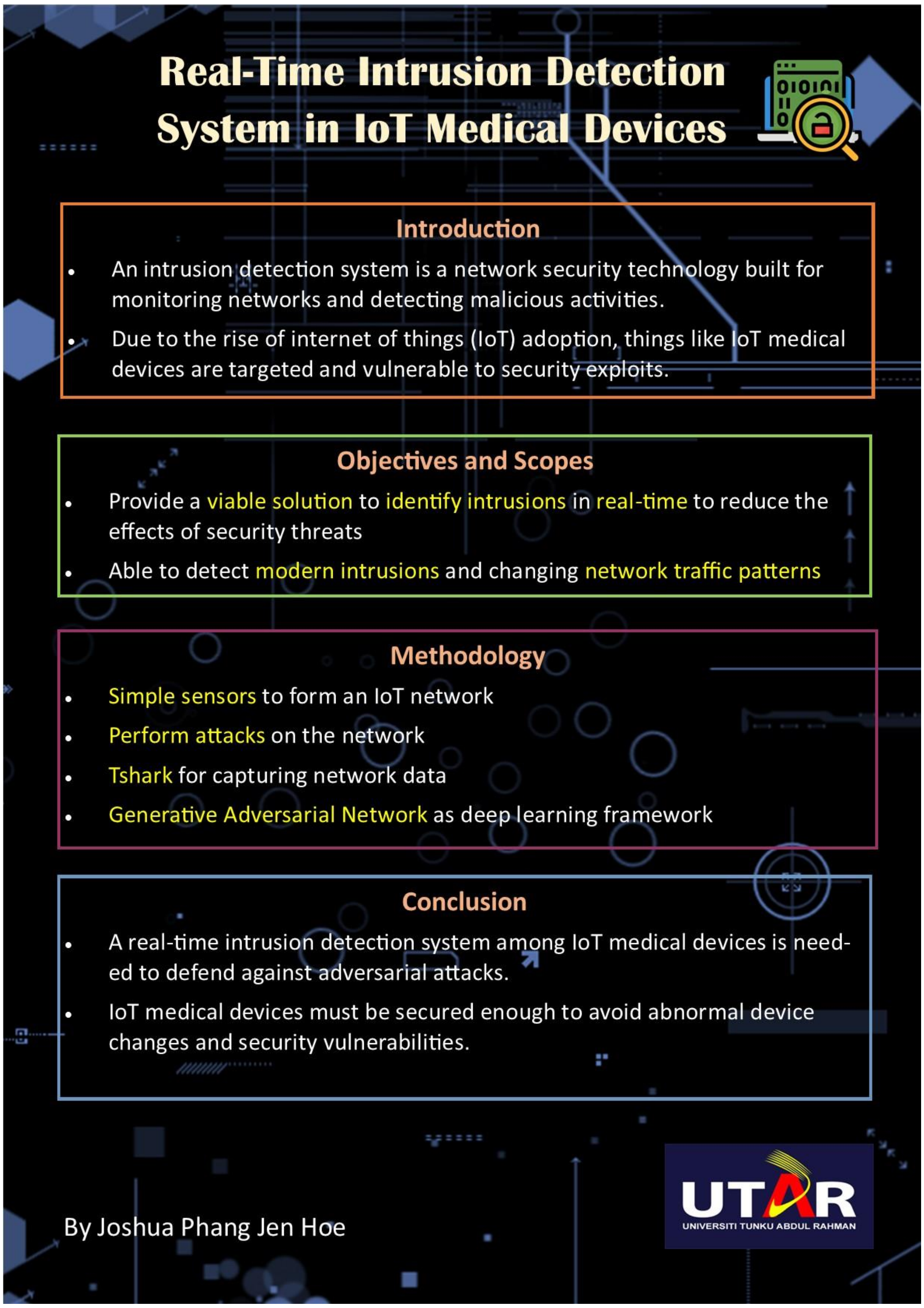
GML

Supervisor's signature

[illegible]

Student's signature

Poster



Real-Time Intrusion Detection System in IoT Medical Devices

Introduction

- An intrusion detection system is a network security technology built for monitoring networks and detecting malicious activities.
- Due to the rise of internet of things (IoT) adoption, things like IoT medical devices are targeted and vulnerable to security exploits.

Objectives and Scopes

- Provide a viable solution to identify intrusions in real-time to reduce the effects of security threats
- Able to detect modern intrusions and changing network traffic patterns

Methodology

- Simple sensors to form an IoT network
- Perform attacks on the network
- Tshark for capturing network data
- Generative Adversarial Network as deep learning framework

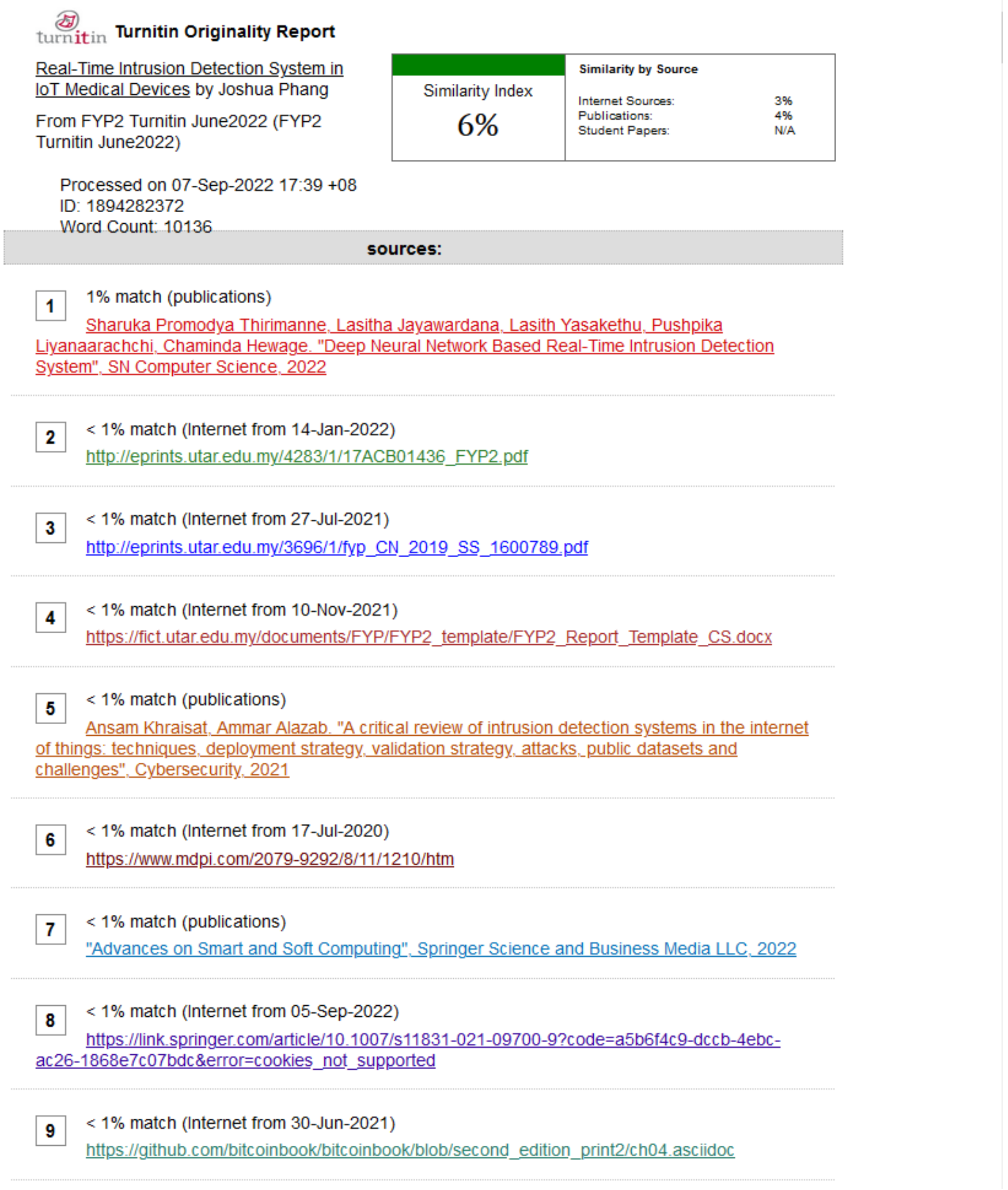
Conclusion

- A real-time intrusion detection system among IoT medical devices is needed to defend against adversarial attacks.
- IoT medical devices must be secured enough to avoid abnormal device changes and security vulnerabilities.

By Joshua Phang Jen Hoe

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Plagiarism Check Results



Turnitin Originality Report

Real-Time Intrusion Detection System in IoT Medical Devices by Joshua Phang

From FYP2 Turnitin June2022 (FYP2 Turnitin June2022)

Processed on 07-Sep-2022 17:39 +08
ID: 1894282372
Word Count: 10136

Similarity Index	Similarity by Source	
6%	Internet Sources:	3%
	Publications:	4%
	Student Papers:	N/A

sources:

1. 1% match (publications)
Sharuka Promodya Thirimanne, Lasitha Jayawardana, Lasith Yasakethu, Pushpika Liyanaarachchi, Chaminda Hewage. "Deep Neural Network Based Real-Time Intrusion Detection System", SN Computer Science, 2022

2. < 1% match (Internet from 14-Jan-2022)
http://eprints.utar.edu.my/4283/1/17ACB01436_FYP2.pdf

3. < 1% match (Internet from 27-Jul-2021)
http://eprints.utar.edu.my/3696/1/fyp_CN_2019_SS_1600789.pdf

4. < 1% match (Internet from 10-Nov-2021)
https://fict.utar.edu.my/documents/FYP/FYP2_template/FYP2_Report_Template_CS.docx

5. < 1% match (publications)
Ansam Khraisat, Ammar Alazab. "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges", Cybersecurity, 2021

6. < 1% match (Internet from 17-Jul-2020)
https://www.mdpi.com/2079-9292/8/11/1210/htm

7. < 1% match (publications)
"Advances on Smart and Soft Computing", Springer Science and Business Media LLC, 2022

8. < 1% match (Internet from 05-Sep-2022)
https://link.springer.com/article/10.1007/s11831-021-09700-9?code=a5b6f4c9-dccb-4ebc-ac26-1868e7c07bdc&error=cookies_not_supported

9. < 1% match (Internet from 30-Jun-2021)
https://github.com/bitcoinbook/bitcoinbook/blob/second_edition_print2/ch04.asciidoc

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Joshua Phang Jen Hoe
ID Number(s)	18ACB06775
Programme / Course	CN
Title of Final Year Project	Real-Time Intrusion Detection System in IoT Medical Devices

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 6 **%** **Similarity by source** Internet Sources: 3 % Publications: 4 % Student Papers: N/A %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

GML

Signature of Supervisor

Name: Gan Ming Lee

Date: 8/9/2022

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student ID	18ACB06775
Student Name	Joshua Phang Jen Hoe
Supervisor Name	Dr Gan Ming Lee

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Plastic Cover (for hardcopy)
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgment
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result – Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]

(Signature of Student)
Date: 7 September 2022