Dissertation on

“Framework for Early Cyber Attack Detection using ML Models deployed on Fog Devices”

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS390B – Capstone Project Phase - 2

Submitted by:

Yuktha Poral	**PES1UG20CS520**
Baddela Divya Malika	**PES1UG20CS540**
Kasturi Uday Aditya	**PES1UG20EC092**
Prateek N Kamath	**PES1UG20CS302**

Under the guidance of

Prof. Vadiraja A
Assistant Professor
PES University

August - December 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

‘Framework for Early Cyber Attack Detection using ML Models deployed on Fog Devices’

is a bonafide work carried out by

in partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE20CS390B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period August- December 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature	Signature	Signature
Prof. Vadiraja A	Dr. Mamatha H R	Dr. B K Keshavan
Assistant Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners **Signature with Date**

1. ____________________________ ____________________________

2. ____________________________ ____________________________

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **“Framework for Early Cyber Attack Detection using ML Models deployed on Fog Devices”** has been carried out by us under the guidance of **Prof.** **Vadiraja A**, Assistant Professor and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester August - December 2023. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1UG20CS520	**Yuktha Poral**
PES1UG20CS540	**Baddela Divya Malika**
PES1UG20EC092	**Kasturi Uday Aditya**
PES1UG20CS302	**Prateek N Kamath**

ACKNOWLEDGEMENT

We would like to express my gratitude to Prof. Vadiraja A, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE20CS390B - Capstone Project Phase – 2.

We are grateful to the project coordinator, Dr Priyanka H, for organizing, managing, and helping with the entire process.

We take this opportunity to thank Dr. Mamatha H R, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support We have received from the department. We would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement we have received from my family and friends.

ABSTRACT

The rapid growth of IoT technology has revolutionized human life by inaugurating the concepts of smart devices, smart healthcare, smart industry, smart cities, and smart grids, among others. IoT devices' security has become a serious concern nowadays, especially in the healthcare domain, where recent attacks exposed damaging IoT security vulnerabilities. Traditional network security solutions are well established. However, due to the resource constraints of IoT devices and the distinct behavior of IoT protocols, the existing security mechanisms cannot be deployed directly to secure IoT devices and networks from cyber-attacks. To enhance the level of security for IoT, researchers need IoT-specific tools, methods, and datasets. To address the mentioned problem, we provide a framework for developing IoT context-aware security solutions to detect malicious traffic in IoT use cases.

With the proliferation of Internet of Things (IoT) devices, early detection of cyber attacks and anomalies using signals only available on the device, rather than traditional network traffic, is an emerging area of research. However, the constraints of IoT devices in memory, compute power, and energy makes deploying machine learning models challenging. In this paper, we propose leveraging TinyML techniques to enable real-time detection of cyber attacks directly on resource-constrained IoT devices. Our approach minimizes communication overhead while providing low latency attack alerts. We evaluate multiple model architectures trained with different optimization techniques to balance accuracy, model size, and inference speed. Our work provides insights into model optimization techniques as well as data collection strategies to improve anomaly detection using intrinsic device signals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

Internet of Things (IoT) devices offer users significant advantages, such as ease of use, compactness, and wireless connectivity. They typically include sensors for various parameters and can be controlled through centralized devices like smartphones or PCs. Examples of first-generation wireless sensor network devices include Zolertia Z1, Sky Mote, and T-mote. The newer generation, commonly referred to as IoT devices, are designed for smart home applications, with companies like Amazon incorporating them into products like dash buttons.

Despite their benefits, studies indicate that around 70 percent of these devices have vulnerabilities. Our experiment involves using ESP32, ESP8266, Raspberry Pico, Raspberry Pi4. Due to their low power and limited computational capabilities, these devices are highly susceptible to attacks like wormholes or flooding, which could have severe consequences for the entire network. To mitigate such threats, a power-efficient mechanism is essential, either in the form of hardware or an algorithm. However, hardware-based approaches pose a risk of increased power consumption within the IoT network.

The proliferation of interconnected IoT ecosystems in critical infrastructures has raised the risk of severe cyber attacks. The WannaCry ransomware campaign exploited vulnerabilities in poorly secured IoT devices, causing widespread damage. The Mirai botnet, infecting hundreds of thousands of Internet-connected cameras and DVRs, demonstrated the dangers of unsecured IoT gateways. Security vulnerabilities in medical equipment like insulin pumps highlight life-threatening consequences if such systems are compromised.

Recent cases of large-scale Distributed Denial of Service (DDoS) attacks, utilizing insecure IoT gadgets, highlight the inadequate attention given to security by manufacturers. This underscores the need for a secure framework applicable to all IoT devices, regardless of their manufacturer, to safeguard organizations and individuals relying on these devices for smart homes.

Traditional security analytics face challenges in deeply embedded IoT endpoints due to computational and memory limitations. Centralized Network Intrusion Detection Systems (NIDS) provide delayed insights, impacting the ability to contain threats before significant damage occurs. To address these issues, we propose a fog computing framework that utilizes machine learning-driven behavioral analytics deployed closer to the affected devices. Near-edge fog nodes with storage and computing capabilities.

CHAPTER-2

PROBLEM STATEMENT

In the intricate landscape of low-latency IoT devices with constrained computational capabilities, the challenge of securing these interconnected entities remains a critical focal point. Traditional network security measures, crafted for more conventional computing environments, falter in the face of the distinct vulnerabilities inherent in low-latency IoT devices. The crux of the issue lies in navigating the delicate balance between implementing robust security protocols and respecting the limited computational resources of these devices. As the demand for rapid data processing and real-time responsiveness intensifies, the imperative to fortify the cybersecurity posture of low-latency IoT devices becomes increasingly evident.

One of the paramount challenges is the early detection of cyber threats in an environment characterized by swift and evolving attack vectors. Conventional security approaches struggle to keep pace with the agility of modern cyber threats, leaving these devices susceptible to sophisticated incursions. Moreover, the unique characteristics of IoT devices, marked by diverse communication protocols and a heterogeneous ecosystem, further complicate the security landscape. Adapting and tailoring security measures to the idiosyncrasies of each device demands a nuanced and comprehensive approach. In essence, the existing problems underscore the urgency for innovative cybersecurity solutions that reconcile the intricacies of low-latency IoT environments with the imperative to fortify defenses against emerging cyber threats.

CHAPTER-3

LITERATURE REVIEW

3.1 IoT Architecture Layers

In the realm of IoT architecture, a widely acknowledged model is the three-layer architecture, originally conceptualized at the nascent stage of IoT experimentation. Comprising the Perception, Network, and Application layers, this framework delineates the fundamental components of IoT systems.

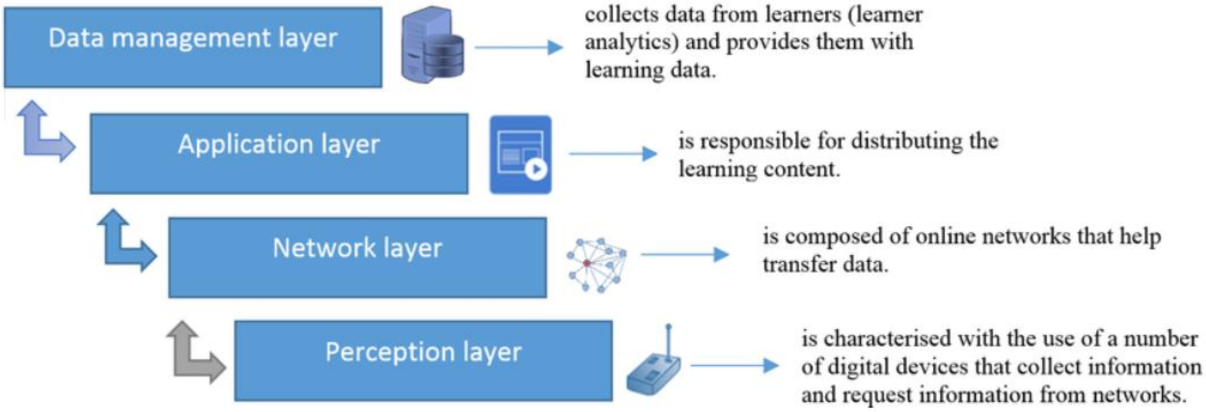

Data management layer
collects data from learners (learner analytics) and provides them with learning data.
Application layer
is responsible for distributing the learning content.
Network layer
is composed of online networks that help transfer data.
Perception layer
is characterised with the use of a number of digital devices that collect information and request information from networks.

Fig 3.1 IoT architecture Layers

The image shows the four layers of a learning system: perception, network, application, and data management.

3.1.1 Perception Layer:

Defined as the physical layer, the Perception layer involves the utilization of sensors, edge devices, and actuators. These components gather diverse data in accordance with project requirements, facilitating interaction with the surrounding environment.

Example: In a smart agriculture system, soil moisture sensors and weather stations serve as components of the Perception layer. These devices collect real-time data on soil conditions and weather patterns, enabling precision irrigation based on environmental needs.

3.1.2 Network Layer:

Tasked with transmitting and processing the collected data, the Network layer serves as the intermediary between devices and other intelligent entities. It establishes connections to smart objects, servers, and network devices, managing the seamless flow of data across the IoT ecosystem.

3.1.3 Application Layer:

The user interface and interaction point reside in the Application layer. This layer caters to specific use-cases of users, exemplified by scenarios such as smart home implementations. Users can, for instance, employ a mobile app to initiate actions like turning on a coffee maker with a simple tap, showcasing the tangible outcomes of IoT functionality.

Example: Within a smart home environment, the Application layer comes to life through user interfaces like mobile apps. Users can interact with the system, such as adjusting thermostat settings or controlling smart lighting, by tapping buttons on their smartphones. This direct interaction exemplifies the tangible benefits of the IoT's Application layer.

3.1.4 Real World Examples :

- **Smart Home:**

  ○ Perception Layer:

  Smart thermostats equipped with temperature and occupancy sensors serve as part of the Perception layer. These devices gather data on room temperature and occupancy patterns, adapting heating or cooling settings for energy efficiency.

  ○ Network Layer:

  The Network layer enables communication between smart home devices like thermostats, smart locks, and lighting systems. Through protocols such as Zigbee or Wi-Fi, these devices exchange data, allowing users to control and monitor their home remotely via a centralized smart home hub or mobile app.

  ○ Application Layer:

  A user-friendly mobile app represents the Application layer in a smart home. Users can interact with and control various devices, such as adjusting thermostat settings, locking doors, or scheduling lighting, providing a seamless and personalized smart home experience.

- **Smart City:**

  ○ Perception Layer:

  Surveillance cameras and air quality sensors contribute to the Perception layer in a smart city. These devices collect real-time data on security and environmental conditions, enhancing overall city awareness.

- Network Layer:

  The Network layer facilitates data transmission from surveillance cameras and environmental sensors to centralized control centers. High-speed communication networks ensure quick and reliable data flow, enabling real-time monitoring and response to security events and pollution levels.

- Application Layer:

  Smart city applications allow citizens to access real-time information on traffic, public transportation, and air quality. Through a user-friendly interface, residents can plan routes, check transportation schedules, and stay informed about the city's current status.

- **Smart Government:**

  - Perception Layer:

    IoT-enabled smart meters in government buildings form part of the Perception layer. These devices monitor energy consumption, water usage, and occupancy, providing valuable data for resource management.

  - Network Layer:

    The Network layer ensures connectivity between smart meters, building automation systems, and central government servers. Secure communication protocols enable the seamless transmission of data, allowing authorities to monitor resource usage and optimize efficiency.

- Application Layer:

  Government officials and administrators access a centralized application to analyze data from smart meters, track resource consumption trends, and make informed decisions for sustainable resource management. The Application layer empowers efficient governance through data-driven insights.

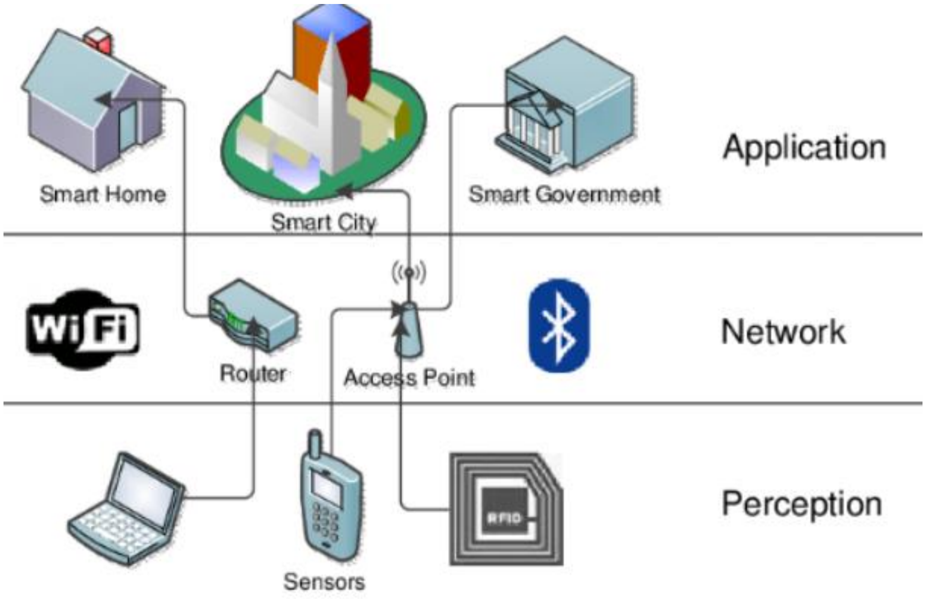


Fig 3.1.4 - 3 Layer Architecture

This diagram shows the three layers of a smart home, smart city, and smart government network, which are connected by a wireless network.

3.2 Cyber Attacks

- Malware:

  Infiltration of IoT devices with harmful software, disrupting operations, stealing data, or gaining unauthorized access by exploiting vulnerabilities in applications.

- Phishing:

  Deceptive tactics trick users into compromising IoT devices, often using fraudulent emails or messages to exploit human behavior.

- Social Engineering:

  Manipulating individuals into revealing sensitive information or granting access to IoT devices through impersonation or psychological tactics.

- Tampering:

  Unauthorized modification of data or configurations on IoT devices, altering settings, manipulating stored information, or injecting malicious code.

- Zero Day Attacks:

  Exploiting unknown vulnerabilities in IoT device applications lacking security updates for unauthorized access or control.

- SQL Injection:

  Inserting malicious SQL code into IoT device input fields, manipulating database queries for unauthorized access or data leakage.

- Man-in-the-Middle:

  Unauthorized interception and modification of communication between devices, compromising data integrity and confidentiality.

- Denial of Service (DoS):

  Overwhelming IoT devices with excessive requests, causing crashes or unresponsiveness, disrupting service availability.

- Distributed Denial of Service (DDoS):

  Multiple devices collaboratively targeting a single device or network, causing a service outage through an overwhelming volume of requests.

- ARP Spoofing:

  Manipulating the Address Resolution Protocol (ARP) table to capture and redirect network traffic, potentially causing unknown parties to gain access.

- DNS Poisoning:

  Modifying the DNS cache to redirect network traffic to malicious sites, compromising the authenticity of IoT device communications.

- Sybil Attack:

  Making many bogus identities to gain unauthorized access or control over IoT devices within the network.

- Rolling Code Attack:

  Intercepting and replaying codes used in keyless entry systems, compromising the security of IoT devices relying on such systems.

- Spoofing:

  Impersonating legitimate devices to gain unauthorized access or control within the IoT ecosystem, potentially leading to security breaches.

- Injection:

  Injecting harmful code or data into IoT device communication streams, resulting in unauthorised access or data modification.

- <u>Replay:</u>

  Capturing and re-transmitting data within the Perception to gain unauthorized access, potentially compromising IoT device security.

- <u>Tampering:</u>

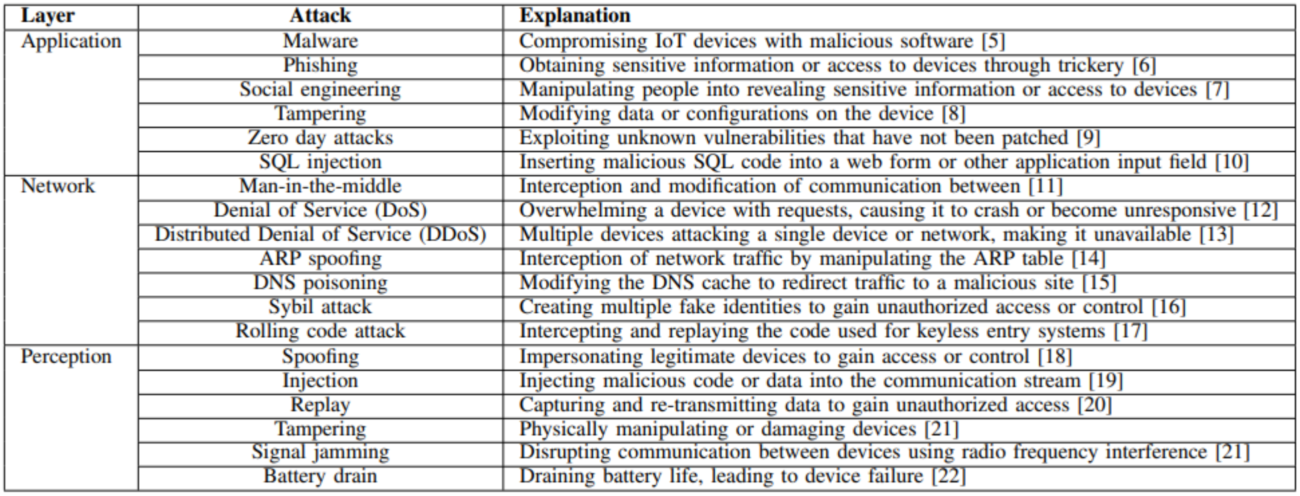
  Physically manipulating or damaging IoT devices, compromising their physical integrity and potentially leading to malfunctions or security breaches.

- <u>Signal Jamming:</u>

  Disrupting communication between IoT devices by introducing radio frequency interference within the Perception, potentially leading to communication failures.

- <u>Battery Drain:</u>

  Intentionally depleting the battery life of IoT devices within the Perception, leading to device failure and potential disruptions in overall system functionality.



Layer	**Attack**	**Explanation**
Application	Malware	Compromising IoT devices with malicious software [5]
	Phishing	Obtaining sensitive information or access to devices through trickery [6]
	Social engineering	Manipulating people into revealing sensitive information or access to devices [7]
	Tampering	Modifying data or configurations on the device [8]
	Zero day attacks	Exploiting unknown vulnerabilities that have not been patched [9]
	SQL injection	Inserting malicious SQL code into a web form or other application input field [10]
Network	Man-in-the-middle	Interception and modification of communication between [11]
	Denial of Service (DoS)	Overwhelming a device with requests, causing it to crash or become unresponsive [12]
	Distributed Denial of Service (DDoS)	Multiple devices attacking a single device or network, making it unavailable [13]
	ARP spoofing	Interception of network traffic by manipulating the ARP table [14]
	DNS poisoning	Modifying the DNS cache to redirect traffic to a malicious site [15]
	Sybil attack	Creating multiple fake identities to gain unauthorized access or control [16]
	Rolling code attack	Intercepting and replaying the code used for keyless entry systems [17]
Perception	Spoofing	Impersonating legitimate devices to gain access or control [18]
	Injection	Injecting malicious code or data into the communication stream [19]
	Replay	Capturing and re-transmitting data to gain unauthorized access [20]
	Tampering	Physically manipulating or damaging devices [21]
	Signal jamming	Disrupting communication between devices using radio frequency interference [21]
	Battery drain	Draining battery life, leading to device failure [22]

Table 3.2 (i) Common IoT security attacks and explanations

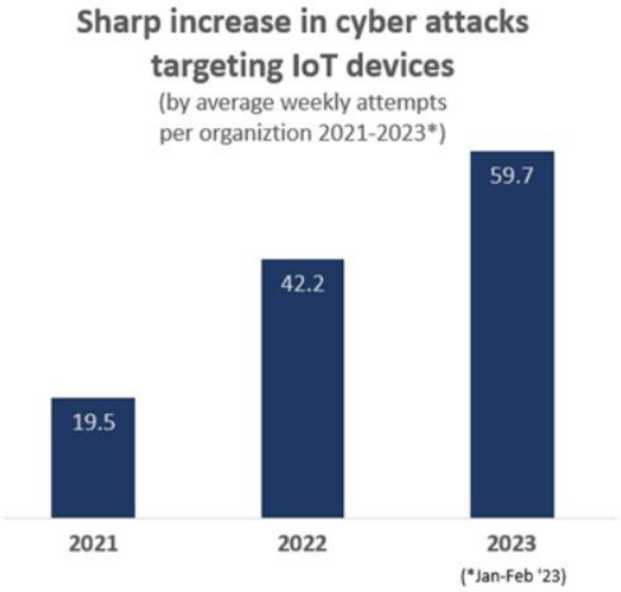


Fig. 3.2.1 Increase in security attacks on IoT devices

The image shows the proliferation in security attacks on IoT devices.

3.3 Machine Learning Algorithms

Depending on their use cases, Machine Learning models are largely classified into two categories:

3.3.1 Supervised ML models

The training dataset is labelled with well defined input-output pairs in this case. The algorithms are taught to translate input parameters to labelled values. It is considered to be supervised since the training data includes the expected outputs [28]. Some algorithms that use this strategy are as follows:

- **Decision Trees:** In [33], Manish Kumar et al. present an Intrusion Detection System based on the DT algorithm. The ID3, C4.5, and C5.0 algorithms were used for experimental analysis on the KDD99 dataset by the authors. This dataset is divided into five categories: normal connections, DoS, R2U, U2R, and probing. ID3, C4.5, and C5.0 all performed well, with Detection Rates of 96.63%, 97.77%, and 98.22%, respectively. They look forward to improving, automating and simplifying this technique.

  [28] gives a brief on decision trees. The model learns basic decision rules to predict output class using the training data. Decision trees are used to forecast the class label of the record, beginning at the root of the tree. The values of the record's attributes are compared to those of the root attribute. Based on the comparison, the next node is searched after the branch that corresponds to that value. This method works for both classification and regression issues. To determine whether or not to split a node, Decision Trees employ a variety of approaches, one of which is the ID3 algorithm.

  The ID3 method generates these Decision Trees by employing a top-down greedy search technique, iterating over each substantially underutilised property in the set S (root node), computing the attribute's information gain (IG) and entropy (H). Entropy E(S) for a single attribute is determined as follows:

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

  Where, pi = Probability of event i in the state S Entropy E(S) fr multiple attributes is given by,

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

T: current state and X: selected attribute.

- **<u>Random Forest</u>**: [34] propose ML methods for the detection of DDoS attacks performed on IoT devices. The authors begin by capturing their own datasets for both benign and attack traffic. These datasets are then used to train ML models such as, Random Forest Classifier, k-Nearest Neighbours, Support Vector Machine, Decision Trees and Neural Networks. All the models have an accuracy of about 99.99%. The models were deployed on a Raspberry Pi 3, which acted as a Wi-Fi access point for IoT devices such as, Smart Home Cameras, Smart Switches, etc

  [30] Random Forest uses several Decision Trees to accomplish classification and regression tasks. It integrates the results of several Decision Trees to produce a single outcome (Ensemble learning approach). First, each decision tree in this model is constructed using a subset of characteristics and data points. m features and n random records are chosen from a dataset of k records. A Decision Tree is constructed for each sample, and the Random Forest model's output is based on the Majority Voting technique for classification problems and Averaging for regression problems.

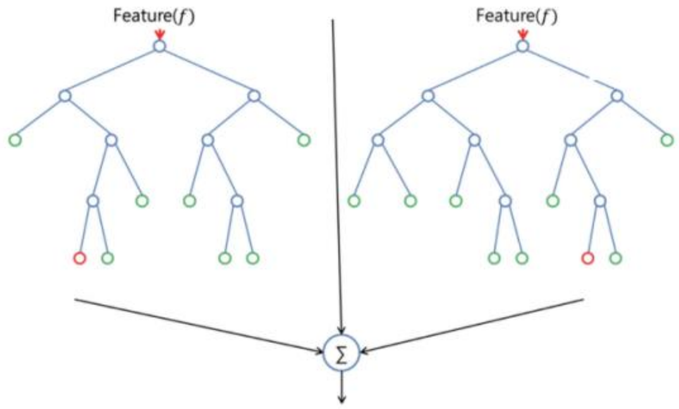


Fig 3.3.1.1: Simplified Overview of a Random Forest model

This image shows a simplified overview of a random forest model, which consists of multiple decision trees that are combined to make predictions.

- **Multilayer Perceptron**: The authors of [35] created a DDoS attack detector based on MLP. The authors create a dataset by scanning and sniffing packets with Wireshark modules, which is then preprocessed and given to a Multilayer Perceptron model for classification. The CiCIDS2017 dataset was used for training purposes. Finally, the MLP model performed admirably, with a classification accuracy of 99.3% and an initial loss value of 0.049 that was later improved to 0.033. Both safe and hostile traffic had accurate rates of 99.9% and 98.3%, respectively.

  [31] discusses in detail about a Multilayer Perceptron. A network of interconnected neurons with several levels. This is a feedforward neural network, which implies that data flows in a single path from the input layer to the output layer without causing cycles. The numerous layers are widely classified into three categories: Input Layer, Hidden Layer(s), and Output

Layers. The input data is transmitted onto each neuron inside the MLP, which is then activated before being passed onto the next neuron. This procedure is repeated until the data reaches the output layer, when it is given the final output. MLP simplifies complicated jobs in this way.

This may be expressed mathematically as,

$$f^{(L)}(x) = a(w_0^{(L)} + \sum_{i=1}^{U_{L-1}} w_i^{(L)} f_i^{(L-1)}(x))$$

L is the layer unit and w represents the weights.

- **Logistic Regression**: Q.A. Al-Haija et al. [36] propose utilising the Logistic Regression Algorithm (LRA) to identify port scan attacks. The authors train the LRA model with the PSA-2017 dataset, which includes 9 features such as time, date, number of frames transmitted per second, number of distinct source and destination MAC addresses per second, throughput (bytes/second), number of distinct source and destination ports per second, and label class. The model was tested with MATLAB 2020b and has an accuracy of 99.4%.

  [31] Briefly discusses what Logistic Regression is. This approach is mostly employed in binary classification. When there are two possible outcomes in a classification job, logistic regression models the likelihood using logistic functions such as a sigmoid curve.

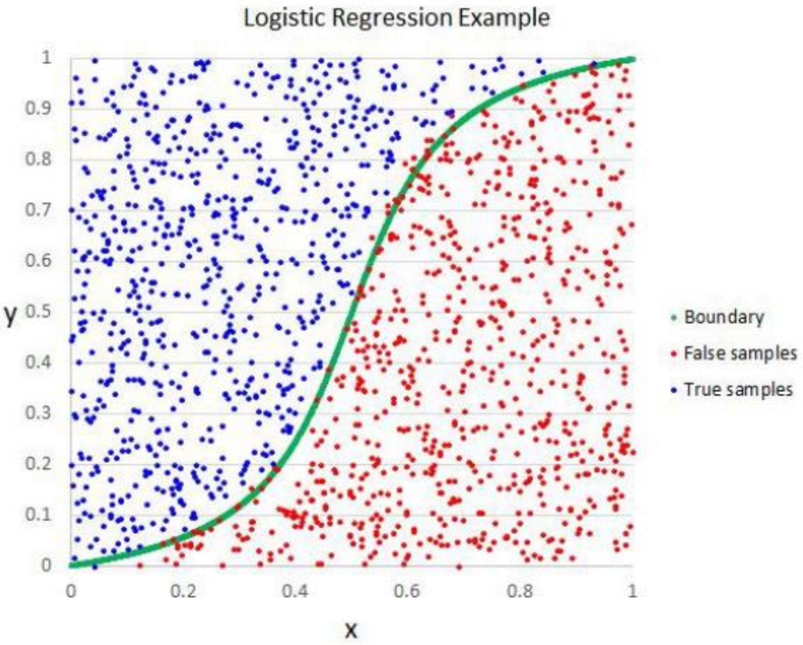


Fig 3.3.1.2: Example of a Logistic Regression Model

This image shows a logistic regression graph, which plots the probability of a binary outcome (e.g., yes/no, pass/fail) versus a predictor variable.

- **<u>k Nearest Neighbours</u>**: In this paper [37], a model based on the K-Nearest Neighbour (KNN) algorithm is suggested for detecting phishing attacks by categorising URLs as malicious or phishing or lawful. The suggested model's performance was evaluated using 106 data to determine its usefulness in detecting phishing. The experimental findings, calculated using accuracy metrics as a performance indicator, show that the suggested model's phishing attack detection capabilities are successful. The suggested model had an overall accuracy rating of 85.08%. Finally, it was concluded that the suggested model is appropriate for dealing with the growing incidence of phishing attacks in the commercial field.

[32] discusses the K-Nearest Neighbour (KNN) algorithm. It is a popular machine learning

approach for classification and regression issues. It is based on the assumption that comparable data points have equivalent labels or values. Throughout the training phase, the KNN algorithm refers to the whole training dataset. Before making predictions, it calculates the distance between each training sample and the input data point using a chosen distance metric, such as Euclidean distance. The algorithm then identifies the K nearest neighbours of the input data point based on their distances. In terms of classification, the method predicts the label for the input data point based on the most common class label among the K neighbours.

- **Support Vector Machine**: SVM (Support Vector Machine) was used in this [38] to classify and forecast SQL-Injection attacks. The suggested approach achieves a detection accuracy of SQL-Injection attacks of 96.47%, which is the highest among current SQL-Injection detection algorithms.

  Support Vector Machine (SVM) is a supervised learning model for regression and classification research [31]. It is highly recommended due to its high accuracy and minimal processing power and complexity. Once the data points are displayed, the Hyperplane can determine if they are linearly or non-linearly separable. SVM handles the case of non-linearly separable data points by mapping them to a higher-dimensional feature space where the data points are linearly separable. SVM guarantees that the data points are at an ideal distance from the Hyperplane in addition to ensuring linear separability.

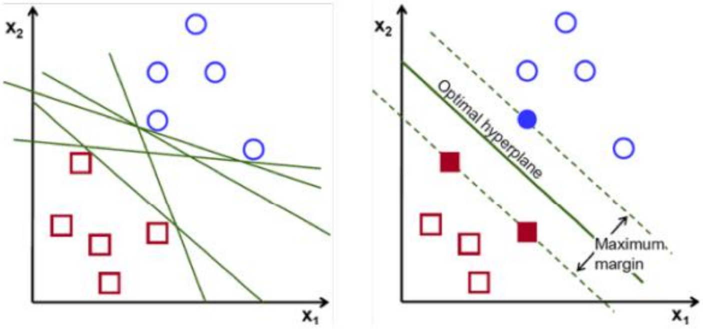


Fig 3.3.1.3: Example of a Support Vector Machine model

The image shows the architecture of a support vector machine (SVM) model, which uses a hyperplane to separate data points into two classes.

3.3.2 Unsupervised ML models

In contrast to Supervised Learning, the data sets are unlabeled, and predictions are formed based on patterns discovered in the dataset. The desired outcomes are not included in the input data in this case.

- **Autoencoder**: In [39], an Autoencoder model was used to detect DDoS attacks. The model had a Detection Rate of 82%.

  [33]Autoencoders are made up of an encoder and a decoder. The encoder converts the input data to feature space, while the decoder converts the data from feature space to data space. Between these two layers is a bottleneck layer, which contains compressed forms of the incoming data.

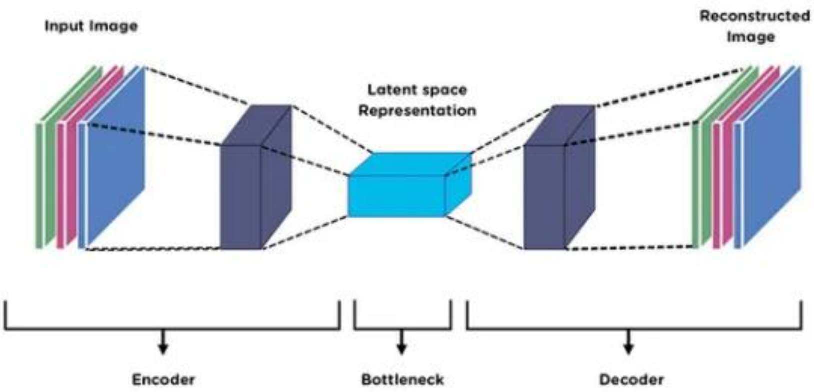


Fig 3.3.2.1: Architecture of the Autoencoder model

This image shows the flow of data through a neural network, which consists of multiple layers of interconnected nodes.

3.4 Levels of Computing in IoT

3.4.1 Edge Computing:

Edge computing stands as a cutting-edge computing paradigm that redefines traditional computational frameworks by relocating processing capabilities to the network's periphery. At its core, edge computing is driven by the strategic imperative to execute computations in close proximity to the source of data, signifying a departure from centralized cloud-centric models [1]. This shift holds profound implications for the architecture of computing systems, offering a responsive and decentralized approach to meet the evolving demands of diverse industries.Notably, the concept of edge computing assumes varying definitions across research perspectives. Pioneered by Shi et al. [2-4], their characterization frames edge computing as a distinctive style of network edge execution. In this formulation, the downlink data of edge computing corresponds to cloud services, the uplink data mirrors the Internet of Everything, and the term "edge" encapsulates the expansive realm of computing and network resources situated between the data source and the trajectory leading to the cloud computing center. This nuanced interpretation underscores edge computing as an intricate interplay between data, computation, and network resources, strategically positioned to enhance efficiency and address the latency challenges inherent in conventional cloud computing models.

As industries increasingly grapple with the imperative for agile linking, real-time business operations, data optimization, application intelligence, and stringent security and privacy considerations, edge computing emerges as a pivotal solution. Its ability to meet the critical requirements for low latency and high bandwidth positions edge computing as a dynamic and integral component of contemporary computing architectures. Consequently, ongoing research endeavors underscore the significance of edge computing as a transformative force, shaping the trajectory of computing paradigms in the digital era.

3.4.2 Fog Computing:

The term "fog" is derived from meteorology and refers to a cloud near the ground. In computing, fog represents the concentration of computational activity near the network's edge. The word is widely linked with Cisco and is thought to have been created by Ginny Nichols, a product line manager at the business. While Cisco Fog Computing is a registered trademark, the notion of fog computing is available to the general public.

While edge devices and sensors play a pivotal role in data generation and collection, their occasional limitation in processing and storage capacity hinders the execution of intricate analytics and machine learning tasks. Conversely, cloud servers possess the capability to undertake these tasks, yet their geographic distance often results in delays in processing data and responding promptly.

Fog computing emerges as a solution to bridge this gap. By strategically situating computational resources between the edge devices and the distant cloud servers, fog computing facilitates short-term analytics at the edge. This allows for swift decision-making and responsiveness, addressing the inherent challenges posed by the processing limitations of edge devices and the geographic constraints of cloud servers. In essence, fog computing acts as a vital intermediary, optimizing the balance between local processing and cloud capabilities to ensure efficient and timely execution of analytics and machine learning tasks.

3.4.3 Cloud Computing:

Cloud computing represents a revolutionary paradigm in which a spectrum of computing resources, encompassing data storage, processing power, and applications, is seamlessly delivered over the internet. Departing from the confines of local devices, data processing and storage now find a dynamic home in remote servers. This decentralization not only amplifies scalability and accessibility but also introduces nuanced challenges concerning latency and responsiveness.

Despite the pervasive reach and computational prowess of cloud servers, a significant drawback emerges from their inherent geographical separation from end-users and edge devices. The spatial divide introduces latency, leading to delays, particularly in scenarios necessitating rapid data processing and real-time responsiveness. This challenge assumes critical importance in applications where low latency is paramount, such as interactive web services, video streaming, or real-time analytics.

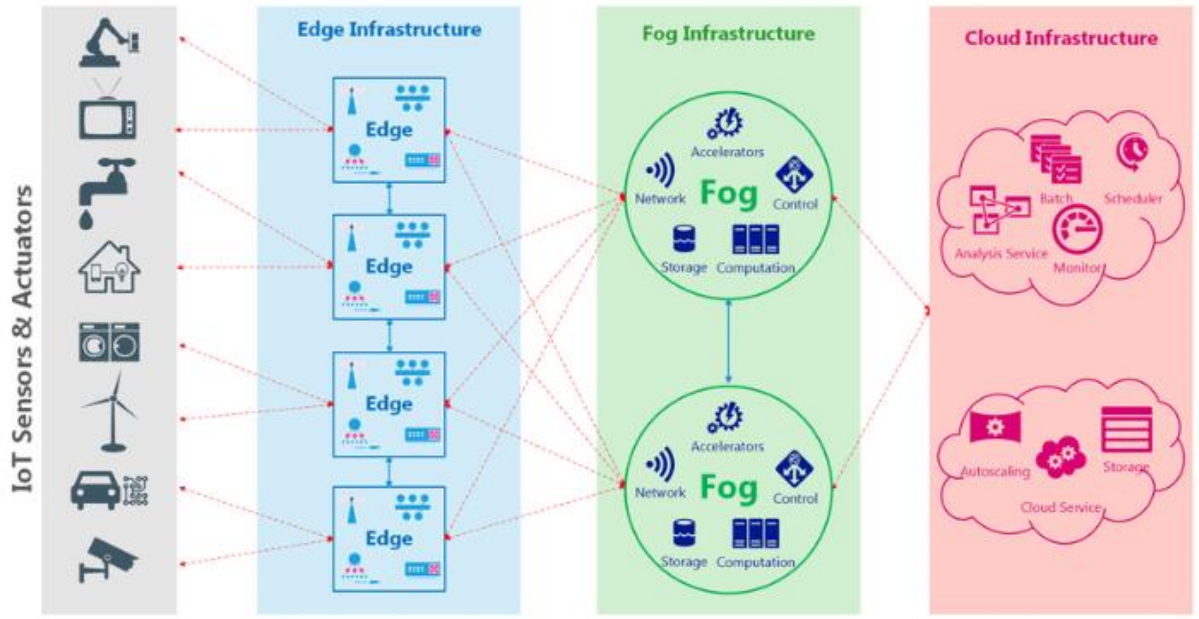


Fig 3.4.3 IIoT edge-fog-cloud architecture

This image shows the proposed edge-fog-cloud architecture, which consists of three layers: edge infrastructure, fog infrastructure, and cloud infrastructure.

3.5 Existing Frameworks for IoT Security

[23] suggests a biometric security architecture for IoT-enabled healthcare that is ML based.The Electrocardiogram (ECG) of a patient is used to extract features for training, and the produced unique biometric EIs from the ECG scan, along with the coefficients from polynomial approximation, are used to verify a user for testing purposes. The authors' primary objective was to safeguard patient medical data by creating a novel framework that, when implemented on medical equipment with limited resources, consumes the fewest resources possible. The multilayer perceptron is employed by the framework.

The goal of [24] is establishing a framework for network-based intrusion detection and mitigation, or IoT-IDM, in smart home settings. This is achieved by using the programmability and network visibility of the SDN architecture in IoT-IDM to provide a more dependable and secure environment for smart homes. Machine learning algorithms are employed by IoT-IDM to detect compromised hosts that are prepared to launch an attack. The attack source is identified, the required rules are developed, and they are applied to underlying routers and switches in order to stop attacks on susceptible IoT devices. The usage of an IoT-IDM prototype, which is currently being developed as a module of the Floodlight SDN controller, is demonstrated using a real smart lighting system. The authors employed a 100% accurate Support Vector Machine model in addition to a 96.2% accurate Logistic Regression approach.

This study [25] provided an Internet of Things (IoT) security framework for smart infrastructures that is divided into four tiers: the network, service, application, and device (end node) layers. The use of the threat model for sensor security and protection in smart IoT infrastructure was shown in the experimental findings. The ability of the Anomaly Behaviour Analysis (ABA) method to identify known as well as unknown threats was demonstrated, with low false positive alarm rates (about 4.2%), and high detection rates. A system was created for classifying attacks, and it was able to classify known attacks with an accuracy of 98% and unknown attacks (referred to as "new attacks") with an accuracy of up to 95%. Currently being worked on is the extending of the ABA technique to the other layers of the IoT security architecture.

[26] explains a real-time attack traffic detection security framework. The Framework is trained on a variety of models, including Gaussian Naive Bayes, k-Nearest Neighbours, Recurrent Neural Networks, and Convolutional Neural Networks. It is then implemented on

Internet of Things devices, including temperature controllers and fire detectors, and it offers a real-time classification of benign and attack traffic. In the end, the scientists concluded that the kNN model performed the best since it required significantly less time to compute than the other models and didn't strain the devices' limited resources. The other models' accuracy ranged from 78 to 99.84%, while the kNN model's accuracy was 96.67%.

[27] Using Network Function Virtualization and Software Defined Networks, a security architecture for Internet of Things systems was put into place. By comparing their method to earlier implementations, the authors were able to use the JRip Classifier to attain an accuracy of 98.7-99.9%.

CHAPTER - 4

SYSTEM REQUIREMENT SPECIFICATION

4.1 Introduction

In the future, the Internet of Things (IoT) will connect many aspects of our life, including smart buildings, smart homes, and even entire cities. A portion of this expansion can be ascribed to advances in the semiconductor sector. IoT will provide cutting-edge services that will necessitate the use of several data centers and real-time processing of all of them. However, technological advancements promise an increase in the attack surface and the risks that these gadgets will confront. IoT security is a developing topic that is growing awareness about the necessity for security methods for these limited devices. Existing solutions in the Fog are computationally demanding for a typical limited device.

With these devices' numerous implementations, there is a need for a security solution that can be scaled to the vast majority of IoT applications. We propose a Python-based Framework comprising several security procedures and functions for real-time detection of Cyber Attacks in their early phases using numerous ML models. This Framework is tested on a device put in the fog, which is linked to an IoT device network. This Framework's unique characteristic is that it takes a dual approach, both from the standpoint of developers and ordinary users.

The functional requirements will offer an overview of the Framework's capabilities, while the non-functional requirements will explain the Framework's need for extensibility.
This Framework's conception, development, and implementation were iterative processes, and it has the ability to avert excessive harm caused by network attacks by detecting it quickly.

4.1.1 Project Scope

The Framework that is being proposed will address the security concerns of IoT devices and will run on a device placed in the Fog layer. This device (Raspberry Pi 4), will behave as a gateway for an IoT network and will constantly monitor the network traffic for any malicious patterns.

The aforementioned framework will provide a real-time detection of attacks such as DoS, DDoS, ARP Spoofing, Botnet, and Phishing. With the help of tools such as Scapy and Pyshark, network traffic is constantly sniffed, collected, labelled, and stored in a Comma Separated Values (.csv) file. The pre-trained ML models are given this csv file as an input and as soon as predictions are made, the sniffing and collection of data begins again. This runs continuously until the user provides an interruption.

Furthermore, the vast majority of IoT devices are barebones, meaning they lack an operating system and only have a few supporting libraries. This necessitates security solutions that are portable from device to device. At the moment, the Framework's capabilities are restricted to slightly more capable devices like the Raspberry Pi. As is customary with technological advancement, new attack vectors will emerge that the Framework will be unable to detect.

4.2 Product Perspective

The advent of IoT devices is partly due to their timid size and extremely malleable applications. IoT devices come in all shapes and forms but they all have one thing in common and that is that they are constrained by their resources. Limited memory and computational capabilities make it difficult for the implementation of traditional security solutions as they prove to be too bulky for these constrained devices. Existing solutions for IoT security are far and few in between, with a focus mainly on Software Defined Networks. Existing works do not provide a real-time analysis.

The proposed Framework works to address these issues. It is a solution to an ever growing problem. The main aim of the Framework was to provide a dual-approach to it, i.e., a developer's and general users' perspective. A developer will need for the Framework to better fit their applications and this Framework gives the developer the autonomy to tweak a vast majority of parameters. For a general user, the Framework includes default datasets and pre-trained models for ease-of-access.

4.2.1 Product Features

- **Packet Sniffing:** Scapy and Pyshark, capture packets in real time. In case of detection of DoS and DDoS attacks, a batch of 10000 packets are captured and sent for further computations.
- **Data Pre-processing:** The captured packets are parsed through and necessary fields are extracted and stored in a csv file. The data in the csv file is encoded, before being sent to the ML models for detection. Each attack has a need for different fields and this has been specified in each function.
- **Prediction:** The ML models will perform computations on the given csv file and provide a detailed report of the prediction in terms of several metrics along with the accuracy.
- **Publishing Results:** In case SYN DoS and DDoS attacks, there is an added functionality to publish the results via MQTT. A receiver may subscribe to this topic for viewing the results.
- **Training and Testing:** Along with default models, the Framework also gives the autonomy for the users to capture their own data and train the models as per their applications. This functionality also allows them to change the various parameters for training and testing these models.

4.2.2 Operating Environment

The Framework presently runs on a Raspberry Pi, i.e., Pi OS. In the architecture, the Framework is situated in the Fog.

4.2.3 General Constraints, Assumptions and Dependencies

Constraints:

- The framework is unable to run directly on edge devices.
- There is an overfitting on the ML models.
- New attack vectors may not be detected.

Assumptions:

- The Framework is running in a place that has consistent network flow.
- The testbed setup is safe.

Dependencies:

- Constant access to the internet.

4.2.4 Risks

Despite the numerous efforts taken to defend the system and maintain its integrity, cyber attacks remain a possibility. These attacks might originate both within and externally. Malicious insiders, angry staff, or rogue gadgets can all be sources of internal risks. External threats might come from organized cyber-attack groups, nation-states, or malicious individuals.

To address these dangers, powerful machine learning models (ML models) on Fog devices for early detection of cyber threats are critical. These ML models can efficiently analyze and classify network data by using the processing power and storage capacity of Fog devices, enabling for fast detection and mitigation of possible attacks.

However, there are several drawbacks to using ML models for early detection of cyber attacks. These models may not be able to detect all forms of attacks, such as complicated attacks on specific systems or network components.

4.3 Functional Requirements

- **Secure communication:** To guarantee that data is exchanged safely between IoT devices and the system, the system should offer secure communication channels.

- **Threat detection and prevention:** To recognize and stop prospective attacks, the system should have procedures in place for threat detection and prevention. Implementing firewalls, intrusion detection systems, and antivirus software falls under this category.

- **Continuous Monitoring and post-detection alerts:** To ensure continuous network traffic monitoring in order to enable real-time detection. If an attack is detected, an alert should be sent to the user right away.

- **Data protection:** To guarantee that sensitive data is safeguarded, the system should employ data protection procedures. Implementing data encryption methods, access control systems, and backup and recovery programs are all part of this.

- **Machine learning models:** The system's ML/DL models should be strong and resistant to possible intrusions. Making sure the models are accurate and impartial, entails putting approaches like model validation, model selection, and bias detection into practice.

- **Plan for handling incidents:** A plan for handling events or breaches involving potential security should be in place. Defining the processes for locating, containing, and minimizing security events is part of this.

4.4 External Interface Requirements

4.4.1 Hardware Requirements:

Raspberry Pi 4: Hosts the Framework and behaves as the gateway for an IoT network.

ESP32, Raspberry Pi Pico W: Comprise the Edge network.

4.4.2 Software Requirements:

Operating System: Pi OS

Libraries and functions used:
Here are the library names from the provided list:

1. 'pyshark': A network packet capture and analysis tool.
2. 'time': Used for time-related tasks like monitoring durations and delays.
3. 'csv': Used to read and write CSV files.
4. 'Pandas': A data manipulation and analysis tool.
5. 'datetime': A term used to work with dates and times.
6. 'signal': A signal is used for signal handling, such as recording the interrupt signal (SIGINT) for graceful termination.
7. 'sklearn': A machine learning library for training, evaluating, and predicting models.
8. 'KNeighborsClassifier': A machine learning algorithm for k-nearest neighbours classification.
9. 'LabelEncoder': A programme that converts non-numeric input into numeric data.
10. 'StandardScaler': This class is used to scale features in machine learning models.
11. 'classification_report' and 'confusion_matrix': Functions for assessing the performance of classification models.
12. 'joblib' is used to save and load machine learning models.
13. 'numpy': Used in Python for numerical computations.
14. 'sys': Access to variables used or maintained by the Python interpreter.

15. 'paho.mqtt.client': MQTT client used to interface with the MQTT (Message Queuing Telemetry Transport) protocol.
16. 'scapy.all' is a packet modification and crafting command.
17.'socket': A low-level networking interface that allows for communication.
18. 'ipaddress': This command provides tools for working with IP addresses.

4.5 Non Functional Requirements

4.5.1. Performance Requirement

- **Efficiency:** The framework must provide accurate results while consuming few resources.
- **Scalability:** The framework should be extendable for utilisation on a variety of devices.

4.5.2 Safety Requirements

We assume that the testbed setup we will utilize is safe.

4.5.3 Security Requirements

- **Data protection:** To guarantee that sensitive data is safeguarded, the project must abide by strong data privacy rules. In order to restrict access to data, this also entails putting access control measures in place.

- **Strong authentication and authorization** controls should be in place in the system to guarantee that only users who have been given permission may access it.

- **Continuous monitoring** is necessary to spot any unusual behavior or unauthorized access attempts in the system. Implementing security event monitoring tools and intrusion detection systems is part of this.

CHAPTER - 5

SYSTEM DESIGN

5.1 Goal

The goal of the project is to develop an extendable Python framework deployed on fog devices for the early detection of cyber attacks. The framework should provide modularity, scalability, and interoperability, allowing users to seamlessly integrate new detection modules, ensuring low-latency response times, resource efficiency, and effective communication with external cybersecurity databases.

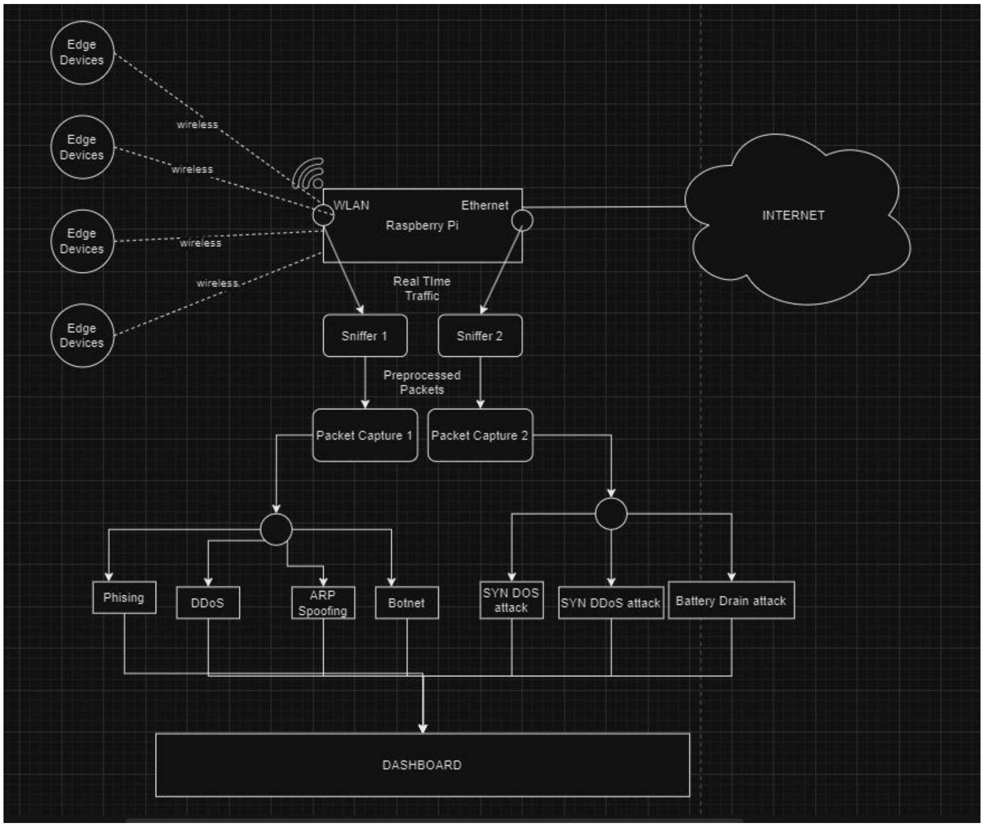


Fig No. 5.1 Proposed methodology

This image shows a network of edge devices, fog devices, and cloud servers, connected by wireless and Ethernet networks, that is used to collect, preprocess, and analyze real-time traffic in order to detect and prevent attacks.

5.2 High Level Design (HLD):

5.2.1 Modular Architecture:

- **Botnet Detection Module:**
  Purpose: Identifies potential Botnet activities within the network.

- **ARP Spoofing Detection Module:**
  Purpose: Detects and prevents ARP Spoofing attacks.

- **DoS and DDoS Analysis Module:**
  Purpose: Analyzes network traffic patterns to detect DoS and DDoS attacks.

- **Phishing Website Detection Module:**
  Purpose: Identifies and blocks phishing websites in real-time.

- **Real-time network traffic processing module:**
  Purpose: Provides a real-time processing engine for efficient and immediate threat detection.

5.2.2 Logs Creation :

- Zeek Execution involves deploying the Zeek network security monitoring tool on a specified interface to capture network traffic, generating detailed log files containing pertinent information about network events.

- Log Processing continuously monitors Zeek-generated log files, identifying new data, and processing it into a structured format like CSV.

5.2.3 Data Storage:

- The communication between the fog and edge devices is encrypted and stored in MongoDB.
- This storage mechanism serves as a foundation for rendering the data within the application layer, ensuring a robust and protected flow of information throughout the edge-to-fog architecture.

5.3 Data Flow Diagram

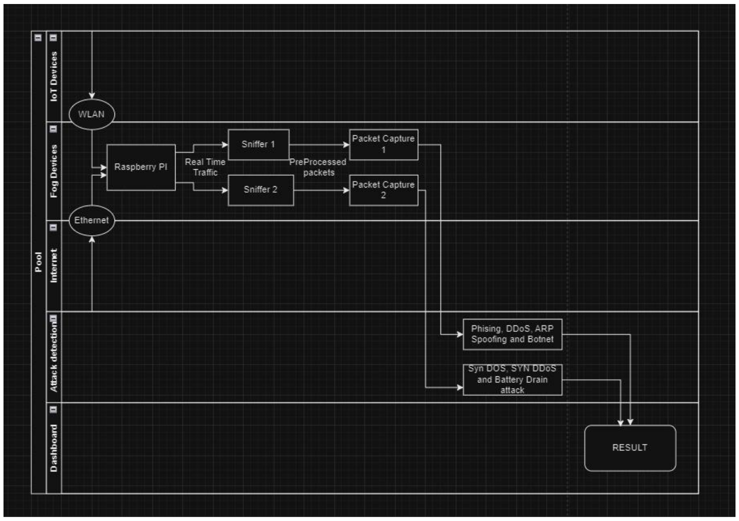


Fig No. 5.3 Data Flow Diagram

This image shows a swimlane diagram to depict the data flow diagram for Framework for early Cyber Attack Detection Using ML Models Deployed On Fog Devices.

5.4 Structure of the library

5.4.1 Botnet attack Detection

- **__init__:** Initializes the BotnetDetector class with various attributes, including the model and dataset splits.
- **print_available_models():** Prints a list of available model names that can be used for training.
- **_preprocess_data():**Standardizes and normalizes the input data for training the model.
- **_split_data():**Splits the dataset into training and testing sets.
- **train_model(dataset, model_name='LogisticRegression'):** Trains a specified model on the given dataset. Supported models include Logistic Regression, Perceptron, Decision Tree Regressor, Gaussian Naive Bayes, K-Nearest Neighbors, and Random Forest Classifier.
- **_preprocess_live_traffic(live_traffic): F**unction for preprocessing live traffic data. Returns the preprocessed data.
- **_predict_default_model(live_traffic):** Predicts using the default trained model. Prints an error message if no model is trained.
- **predict_with_model(live_traffic, model_name='default'):** Predicts using a specified model (or the default model if 'default' is chosen). Supports multiple classifiers.
- **evaluate_all_models(dataset):** Evaluates all available models on the provided dataset. Trains each model and prints the accuracy score.
- **evaluate_model(model=None):** Evaluates a specified model (or the default model if none is provided) on the testing set. Prints the accuracy score.

5.4.2 ARP Spoof Detection

- **__init__:**

  Purpose: Initializes the ArpSpoofDetector object with necessary attributes.

  Attributes: IP_MAC_PAIRS: Dictionary to store IP-MAC pairs.

  ARP_REQ_TABLE: Dictionary to store ARP request information.

  sniff_requests_thread: Thread for sniffing ARP requests.

  sniff_replays_thread: Thread for sniffing ARP replies.

  alarms: List to store alarm messages.

- **start_detection:**

  Purpose: Initiates the ARP spoofing detection by starting sniffing threads.

- **sniff_requests:**

  Purpose: Sniffs outgoing ARP requests.

  Uses Scapy's sniff function with a filter for ARP requests and a callback for handling them.

- **sniff_replays:**

  Purpose: Sniffs incoming ARP replies.

  Uses Scapy's sniff function with a filter for ARP replies and a callback for handling them.

- **print_arp:**

  Purpose: Prints ARP information based on the ARP operation type.

  Prints the source and destination IP and MAC addresses

- **incoming_reply:**

  Purpose: Determines if a packet is an incoming ARP reply.

Returns True if the packet is an ARP reply and the source IP is not the local interface.

- **outgoing_req:**

  Purpose: Determines if a packet is an outgoing ARP request.

  Returns True if the packet is an ARP request and the source IP is the local interface.

- **add_req:**

  Purpose: Adds an ARP request to the ARP request table.

  Records the destination IP and the current timestamp.

- **check_arp_header:**

  Purpose: Checks the consistency of ARP messages.

  Compares source and destination MAC addresses, raising an alarm if inconsistent.

- **known_traffic:**

  Purpose: Handles known ARP traffic.

  Checks if the source IP is known, and if the MAC address matches.

  Raises an alarm if not.

- **spoof_detection:**

  Purpose: Detects ARP spoofing.

  Checks if the ARP request was recently made and verifies the IP-MAC pair. Raises an alarm if fake.

- **alarm:**

  Purpose: Raises an alarm for detected ARP spoofing.

  Adds an alarm message to the list and returns the message.

5.4.3 Phishing

- **__init__(self):** Initializes the PhishingWebsiteDetection class.
  Defines a regular expression for various URL shortening services.

- **explore_data(self, data0):** Displays basic exploration of the dataset.
  Shows the first few rows, shape, columns, info, histogram, correlation heatmap, and description of the dataset.

- **preprocess_data(self, data0):**Handles data preprocessing steps.
  Drops the 'Domain' column, checks for null values, shuffles the data, and returns the processed data.

- **split_data(self, data):** Splits the data into training and testing sets.
  Returns X_train, X_test, y_train, y_test using the train_test_split function.

- **train_decision_tree(self, X_train, X_test, y_train, y_test):**
  Trains a Decision Tree classifier.
  Returns the trained model, and accuracy scores for both training and testing sets.

- **train_random_forest(self,X_train,X_test,y_train,y_test,model_filename="random_forest_model.pkl")**: Trains a Random Forest classifier.
  Saves the trained model as a pickle file. Returns the trained model and accuracy scores for both training and testing sets.
- **train_mlp_classifier(self, X_train, X_test, y_train, y_test)**:
  Trains a Multi-Layer Perceptron (MLP) classifier.
  Returns the trained model and accuracy scores for both training and testing sets.

- **train_autoencoder(self, X_train, X_test)**:
  Trains an autoencoder for feature extraction.
  Returns the trained autoencoder model and accuracy scores for both training and testing sets.

- **train_svm_classifier(self, X_train, X_test, y_train, y_test)**:

Trains a Support Vector Machine (SVM) classifier.
Returns the trained model and accuracy scores for both training and testing sets.

- **display_results(self, model_name, acc_train, acc_test)**:
  Displays the results (accuracy scores) of a trained model.

- **_havingIP(self, url)**: Checks if the URL has an IP address.
  Returns 1 if it has an IP, 0 otherwise.

- **_haveAtSign(self, url)**: Checks if the URL contains an '@' symbol.
  Returns 1 if '@' is present, 0 otherwise.

- **_getLength(self, url)**: Checks the length of the URL.
  Returns 1 if length is greater than 54 characters, 0 otherwise.

- **_getDepth(self, url)**: Computes the depth of the URL path.
  Returns the depth value.

- **_redirection(self, url):** Checks for the presence of additional redirection in the URL.
  Returns 1 if redirection is present, 0 otherwise.

- **_httpDomain(self, url):** Checks if the domain uses HTTPS.
  Returns 1 for HTTPS, 0 otherwise.

- **_tinyURL(self, url):** Checks if the URL is from a known URL shortening service.
  Returns 1 if it is a short URL, 0 otherwise.

- **_prefixSuffix(self, url):** Checks for the presence of '-' in the domain (potential phishing indicator).
  Returns 1 for phishing, 0 for legitimate.

- **_web_traffic(self, url):** Fetches the number of backlinks to the URL.
  Returns 0 for phishing if backlinks are present, 1 otherwise.

- **_domainAge(self, domain_name):** Calculates the age of the domain. Returns 1 if the age is less than 6 months, 0 otherwise.

- **_domainEnd(self, domain_name):**Checks the time remaining until the domain expires. Returns 1 if less than 6 months remaining, 0 otherwise.

- **_iframe(self, response):** Checks for the presence of iframes in the HTML response. Returns 1 if iframes are present, 0 otherwise.

- **_mouseOver(self, response):** Checks for the presence of mouseover events in the HTML response. Returns 1 if mouseover events are present, 0 otherwise.

- **_rightClick(self, response):** Checks for the presence of right-click events in the HTML response. Returns 1 if right-click events are present, 0 otherwise.

- **_forwarding(self, response):** Checks for URL forwarding in the HTTP response history. Returns 1 if forwarding is present, 0 otherwise.

- **featureExtraction(self, url):** Extracts features for a given URL using the previously defined functions. Returns a list of extracted features.

- **load_model(self, filename):** Loads a pre-trained model from a pickle file. Returns the loaded model.

- **predict(self, url, model_name="random_forest_model"):** Generates predictions for a given URL using a specified or default model. Returns the predicted features.

5.4.4 SYN_DoS_DDoS Attack Detection

- **__init__ :** The PacketCapture class is initialized with default or given settings for attributes such iface_name, allowed_IP, and capture_limit. Configures the MQTT client connection details.

- **get_ip_layer_name(packet):** Based on the provided packet, determines and returns the IP layer version (IPv4 or IPv6).

- **save_data(filename='test.csv'):** The collected packet data is saved to a CSV file.

- **interrupt_handler(signal, frame):** Handles interruption signals (e.g., KeyboardInterrupt) by preserving data, publishing capture session information, making predictions, and disconnecting MQTT.

- **publish_results_via_mqtt(message):** Publishes a message to a specified MQTT topic.

- **capture_packets():** PyShark is used to capture network packets and process the information included in each packet. Extracts pertinent information such as source/destination IP addresses, ports, and flags and saves it to a CSV file.
  Packet statistics such as packets per second and delay between packets are calculated. Stops capturing when the specified capture limit is reached.

- **train_capture_packets():** Captures network packets for both benign and malicious traffic. The recorded data is written to CSV files for further training.

- **train_kNN():** Uses the combined data from benign and attack traffic to train a k-Nearest Neighbors (kNN) classifier.Stratified Cross-Validation is used to tune hyperparameters.Evaluation metrics such as the confusion matrix and classification report are printed.
  The trained kNN model is saved to a file.

- **test_kNN():** Loads a previously trained kNN model. The model is tested using attack traffic data and the forecasts are printed. Based on the majority class, determines whether an attack is detected.

- **prediction():** Loads a pre-trained kNN model and applies it to the collected traffic to predict whether it is a SYN DoS attack. The result is published to a MQTT topic.

- **publish_connected_devices():** Publishes device information (IP and MAC addresses) to a MQTT topic.

- **get_local_network():** Gets the IP address of the local network.

- **scan(ip_range):** To detect connected devices, an ARP scan is performed on the specified IP range. The IP and MAC addresses of the identified devices are saved.

- **print_devices():** This command prints information about connected devices.

CHAPTER - 6

PROPOSED METHODOLOGY

6.1 Creation of Hotspot and Real-Time Time Traffic Monitoring

- A pivotal role is assumed by a fog computing device, exemplified by the Raspberry Pi4. Positioned as a central nexus for network communication, this device establishes connectivity to the primary network through a judiciously configured Ethernet cable. Its strategic configuration extends further to the establishment of a wireless hotspot, ensuring seamless connectivity for a spectrum of edge devices, including ESP32 and ESP8266.

- This architectural configuration serves a dual purpose by not only optimizing the deployment and management of devices within a localized environment but also accentuating the critical need for robust security measures and meticulous configuration. Both the fog device gateway and its associated edge devices are subject to a meticulous setup to ensure the establishment of reliable communication pathways. This meticulous approach significantly contributes to enhancing the overall security posture of the network, emphasizing the architecture's commitment to fostering a resilient and well-protected operational environment.

- Beyond its primary function as a communication hub, the fog computing device in this architectural paradigm integrates advanced capabilities for real-time traffic monitoring. This augmentation stands as a critical pillar within the broader security infrastructure, facilitating continuous surveillance of data flows coursing through the network. The seamless integration of monitoring functionalities into the fog computing device exemplifies a proactive stance towards security.

6.2 Telemetry traffic capture and pre-processing

- The architectural framework introduces a dual-interface paradigm, distinguishing between wired (Ethernet) and wireless interfaces. This demarcation underscores the need for a nuanced approach in conducting real-time traffic analysis and preprocessing due to the unique characteristics inherent in each interface.

- The imperative of continuous monitoring is central to real-time traffic analysis on both interfaces. To effectively address the challenges presented by wired and wireless environments, specialized tools such as Zeek, Scapy, Wireshark, and PyShark play a pivotal role in capturing telemetry data. These tools, selected for their adaptability, become indispensable components, ensuring comprehensive and responsive data acquisition.

- At the confluence of these interfaces, preprocessing assumes critical significance, aiming to seamlessly integrate and align diverse data streams with user requirements. This pivotal phase mandates the application of distinct techniques such as protocol normalization and feature extraction. The tailored application of these methods becomes imperative to accommodate the nuanced characteristics inherent in wired and wireless traffic.

- In essence, the meticulous differentiation in the application of preprocessing techniques serves as the linchpin for achieving cohesion within the framework. By addressing the unique attributes of both wired and wireless traffic, the system not only ensures a harmonious convergence but also optimally caters to user requirements. This strategic and differentially applied preprocessing sets the stage for a cybersecurity architecture that is both adaptive and robust in its ability to contend with the intricacies of contemporary network environments.

6.3 Prediction and Framework

- Intricate Framework Development:

  Leveraging intricately preprocessed data, a meticulously crafted framework is deployed to systematically observe and identify anomalies within the network infrastructure.

- Cutting-Edge Algorithms and ML Models:

  The framework integrates state-of-the-art algorithms and machine learning models,ensuring a sophisticated analytical layer when applied to the prepared data. This empowers the system to discern nuanced patterns indicative of irregularities or unexpected behaviors.

- Autonomous Anomaly Flagging:

  The primary objective of this advanced framework is to autonomously flag and bring attention to potential anomalies. It goes beyond mere identification, facilitating prompt and targeted responses to maintain the network's integrity.

- Real-Time Proactive Operation:

  Operating seamlessly in real-time on the fog device gateway, the framework stands as a pinnacle of proactive anomaly detection. Its deployment exemplifies a robust commitment to fortifying the security architecture, staying one step ahead of potential threats through continuous vigilance and analysis.

CHAPTER - 7

IMPLEMENTATION

7.1 Devices In the Network:

7.1.1 ESP32:

Description: The ESP32 is a versatile microcontroller with integrated Wi-Fi and Bluetooth capabilities. It is widely used for IoT applications due to its low power consumption, cost-effectiveness, and compatibility with various sensors and peripherals.

Purpose in the Project: In this cybersecurity framework, ESP32 serves as one of the edge devices responsible for capturing network traffic and forwarding it to the fog gateway (Raspberry Pi 4) for analysis. Its wireless capabilities make it suitable for deployment in diverse environments, and its compact size allows for unobtrusive placement.

Fig No. 7.1.1 ESP 32

7.1.2 Raspberry Pico W:

Description: The Raspberry Pico W is a microcontroller board developed by the Raspberry Pi Foundation. It features the Raspberry Pi-designed RP2040 microcontroller chip and provides a flexible and low-cost solution for embedded systems.

Purpose in the Project: Raspberry Pico W is utilized as another edge device in the network. Its role involves processing and forwarding data to the fog gateway. The cost-effectiveness and programmability of Raspberry Pico make it an ideal choice for lightweight yet efficient tasks in the edge computing domain.

Fig.No. 7.1.2 Raspberry Pico

7.1.3 Raspberry Pi 4:

Description: Raspberry Pi 4 is a single-board computer that offers increased processing power and memory compared to its predecessors. It supports various operating systems and provides connectivity options, including Ethernet and USB ports.

Purpose in the Project: In this framework, Raspberry Pi 4 assumes the role of the fog gateway, responsible for receiving, analyzing, and responding to the network traffic forwarded by the edge devices. Its computational capabilities, connectivity features, and the ability to run complex analyses make it well-suited for acting as an intermediary between the edge devices and the central cybersecurity system.

Each device in this project is selected based on its specific strengths and capabilities, ensuring a well-integrated and efficient system for early cyber attack detection in the network. The combination of ESP32, Raspberry Pico, and Raspberry Pi 4 allows for a distributed and collaborative approach to security monitoring and analysis.

Fig No. 7.1.3 Raspberry Pi 4

7.2 Attacks In Scope

- Phishing
- Botnet
- ARP Spoofing
- SYN DoS and DDoS

7.3 TestBed Setup

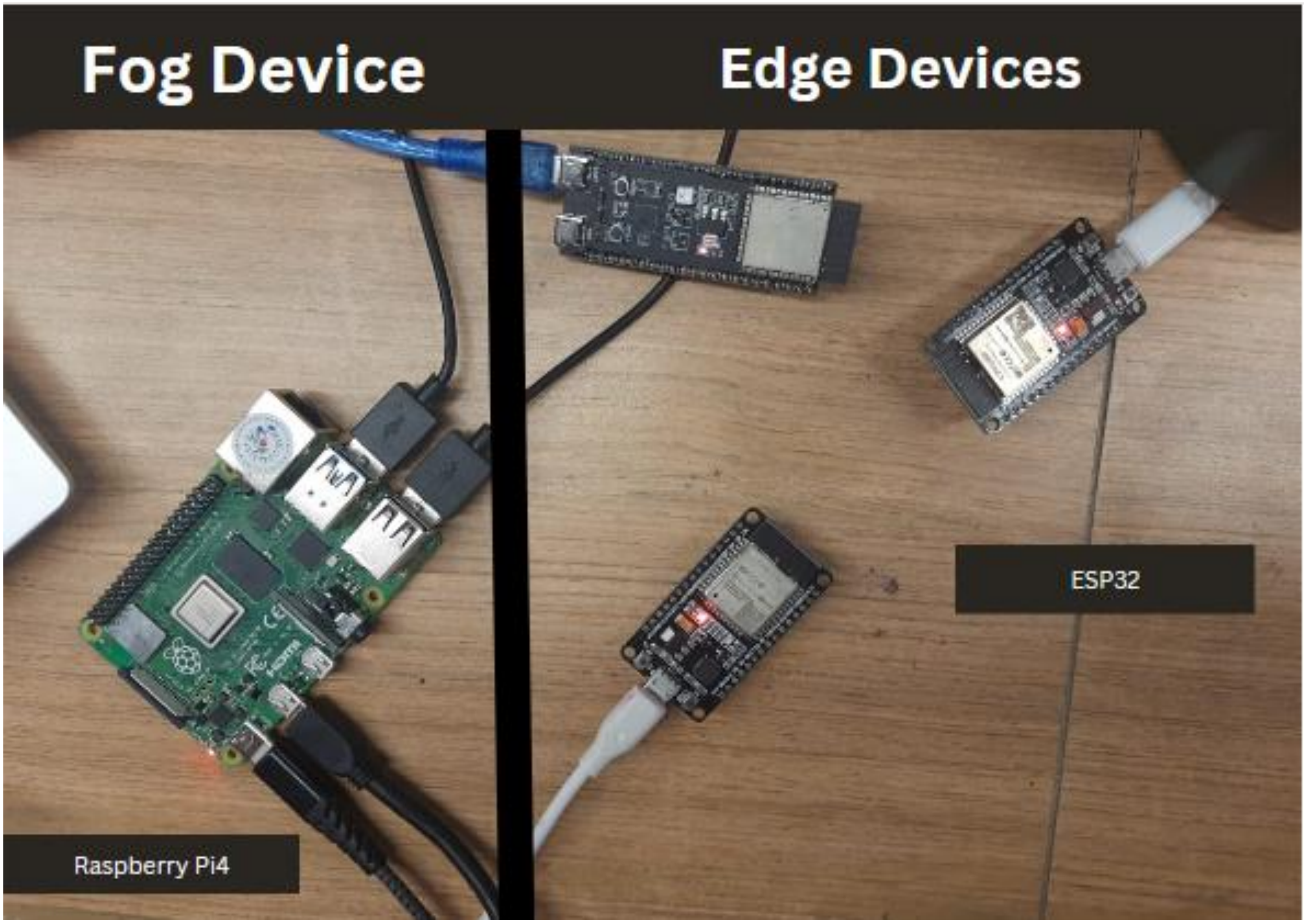


Fig 7.3.1 Test Bed Set up

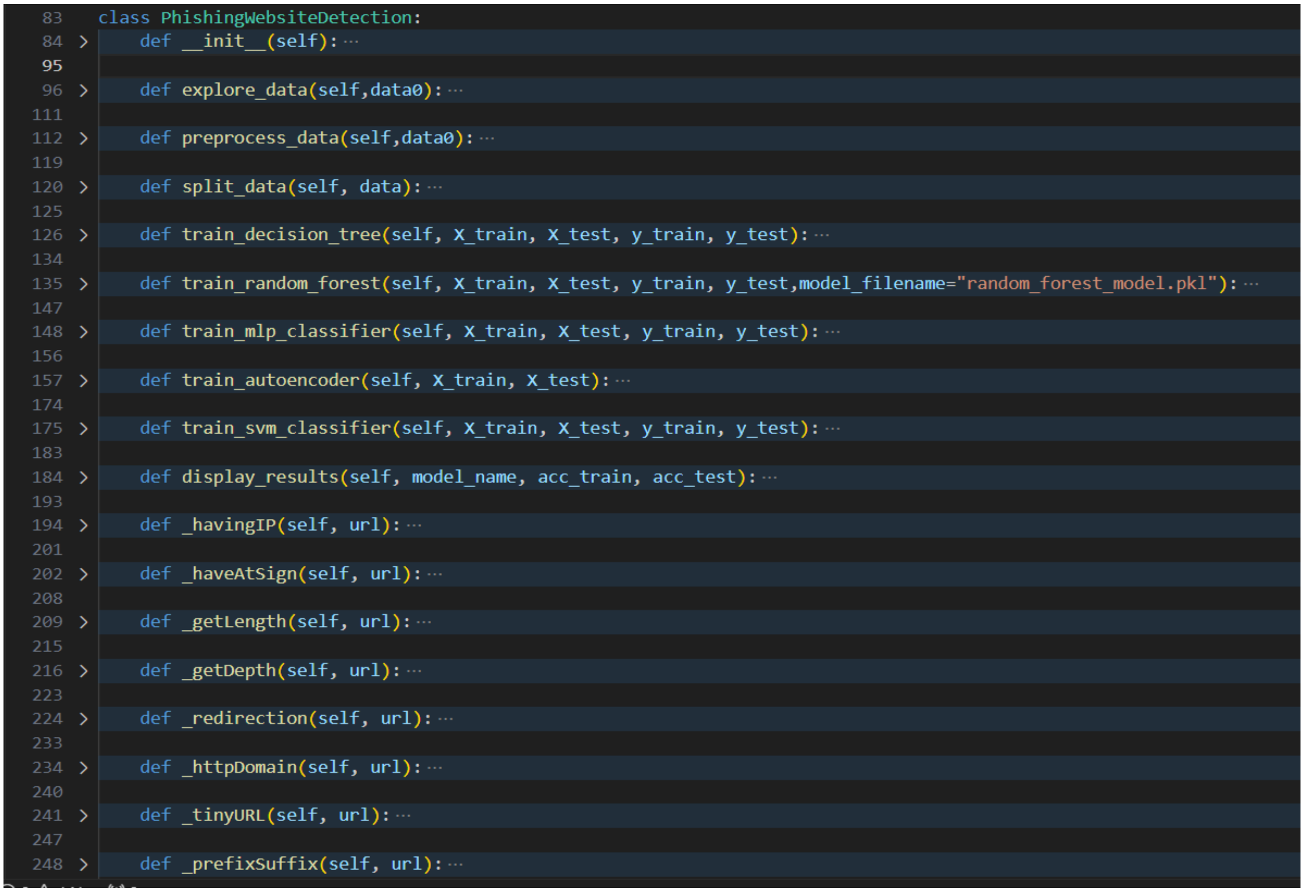
Fig 7.3.1 Shows the test bed with fog device Raspberry Pi4 connected with wired peripheral devices and also edge devices being connected to Raspberry Pi4 via wireless network.

7.4 Phishing Detection class

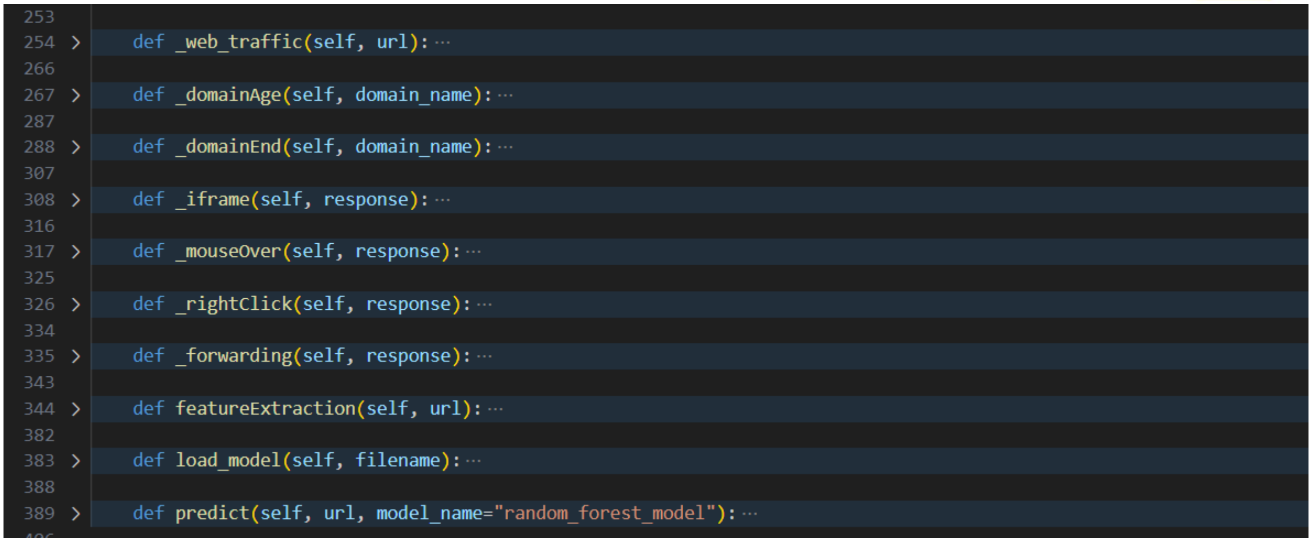
7.4.1 ML Models

- DecisionTreeClassifier
- RandomForestClassifier
- MLP Classifier
- Autoencoder
- SVM Classifier

7.4.2 Interfaces



```
class PhishingWebsiteDetection:
    def __init__(self): ···

    def explore_data(self,data0): ···

    def preprocess_data(self,data0): ···

    def split_data(self, data): ···

    def train_decision_tree(self, X_train, X_test, y_train, y_test): ···

    def train_random_forest(self, X_train, X_test, y_train, y_test,model_filename="random_forest_model.pkl"): ···

    def train_mlp_classifier(self, X_train, X_test, y_train, y_test): ···

    def train_autoencoder(self, X_train, X_test): ···

    def train_svm_classifier(self, X_train, X_test, y_train, y_test): ···

    def display_results(self, model_name, acc_train, acc_test): ···

    def _havingIP(self, url): ···

    def _haveAtSign(self, url): ···

    def _getLength(self, url): ···

    def _getDepth(self, url): ···

    def _redirection(self, url): ···

    def _httpDomain(self, url): ···

    def _tinyURL(self, url): ···

    def _prefixSuffix(self, url): ···
```



```

    def _web_traffic(self, url):···

    def _domainAge(self, domain_name):···

    def _domainEnd(self, domain_name):···

    def _iframe(self, response):···

    def _mouseOver(self, response):···

    def _rightClick(self, response):···

    def _forwarding(self, response):···

    def featureExtraction(self, url):···

    def load_model(self, filename):···

    def predict(self, url, model_name="random_forest_model"):···
```

Fig No. 7.4.2 Phishing detection Class

This image depicts all the functions needed to detect for phishing attack and present in a class.

7.4.3 Default Dataset

- 5000 Legitimate URLs from the University of New Brunswick
- 5000 Phishing URLs from PhishTank

7.4.4 Dataset Features

The following category of features are selected:

- Address Bar-Based Functions
  '//' in URL IP Address in URL 'http/https' in Domain name '@' Symbol in URL Using URL Shortening Service Length of URL Prefix or Suffix "-" in Domain Depth of URL
- Domain-specific Features
- Domain DNS Record Age Website Traffic Domain End Period

- HTML and Javascript-based Functionality

  Redirection of an Iframe

  Disabling Right-click to select

  Customization of the Status Bar

  Website Redirecting

7.4.5 Features Significance

The selected feature categories for URL analysis exhibit a meticulous approach to capturing diverse aspects of web addresses.

Address Bar-based features delve into the intricacies of URL composition, scrutinizing elements such as domain, protocol usage, and the presence of symbols indicative of redirection or email-related patterns. Domain-based features further contribute to a comprehensive understanding of domain-specific attributes.

Simultaneously, HTML & Javascript-based features play a pivotal role in unraveling the structural and behavioral nuances of web content.

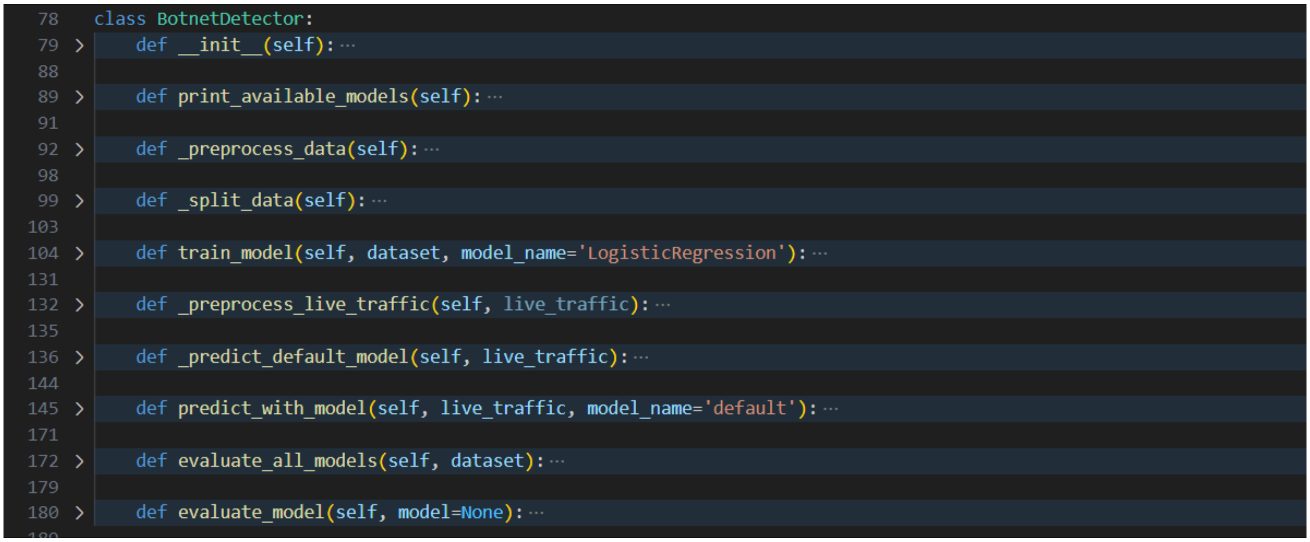
In totality, a meticulous selection process yields 17 distinctive features, each holding strategic significance in discerning the authenticity and potential threats associated with URLs. The integration of these features forms a robust foundation for constructing a model adept at URL categorization and threat detection, offering a nuanced and thorough analysis of web security measures.

7.5 Botnet Detection Class

7.5.1 ML Models

- Logistic Regression
- Perceptron
- Gaussian Naive Bayes
- K-Neighbors Classifier
- Random Forest Classifier
- Decision Tree Regressor

7.5.2 Interfaces



```
class BotnetDetector:
    def __init__(self):···

    def print_available_models(self):···

    def _preprocess_data(self):···

    def _split_data(self):···

    def train_model(self, dataset, model_name='LogisticRegression'):···

    def _preprocess_live_traffic(self, live_traffic):···

    def _predict_default_model(self, live_traffic):···

    def predict_with_model(self, live_traffic, model_name='default'):···

    def evaluate_all_models(self, dataset):···

    def evaluate_model(self, model=None):···
```

Fig No. 7.5.2 Botnet Detection Class

This image depicts all the functions in a class required to detect Botnet attacks

7.5.3 Default Dataset

- CTU-13 dataset (Real botnet traffic mixed with normal traffic)

7.5.4 Dataset Features

- Temporal Attributes:
  - Start time
  - Duration
  -
- Protocol and Port Details:
  - Information about the communication protocols used
  - Specific ports involved
- Source and Destination Addresses:
  - Origin and target of network connections
  - Ports for source addresses
  - Ports for destination addresses
- Direction and State Attributes:
  - Flow
  - Status of traffic
- Type of Service Indicators:
  - Contribute to understanding the nature of communication
- Quantitative Metrics:
  - Total bytes exchanged during communication
  - Total packets involved in the exchange
  - Source bytes quantifying the volume of data exchanged

7.5.5 Features Significance

It encompasses temporal attributes, including start time and duration, providing valuable insights into the temporal aspects of network connections. Detailed protocol and port information furnishes insights into the communication protocols employed and the specific ports engaged.

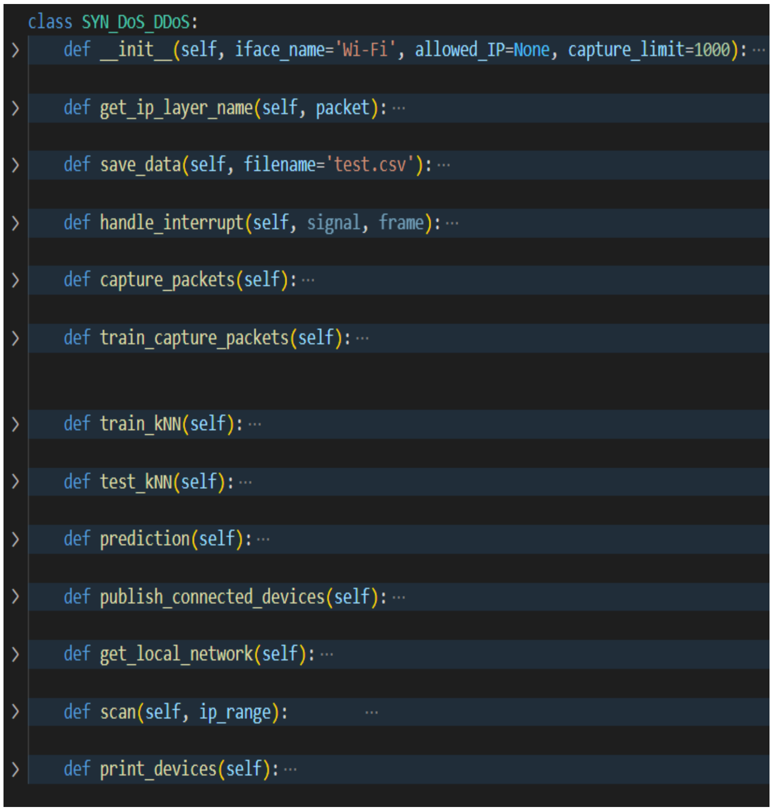
Source and destination addresses, coupled with their corresponding ports, delineate the origin and target of network connections. Direction and state attributes shed light on the flow and status of traffic, while type of service indicators contribute to a nuanced understanding of communication nature. Furthermore, the dataset integrates quantitative metrics such as total bytes, total packets, and source bytes, effectively quantifying the volume of exchanged data.

7.6 SYN DoS and SYN DDoS Detection class

7.6.1 ML Models

- k-Nearest Neighbours

7.6.2 Interfaces



```
class SYN_DoS_DDoS:
    def __init__(self, iface_name='Wi-Fi', allowed_IP=None, capture_limit=1000): ···

    def get_ip_layer_name(self, packet): ···

    def save_data(self, filename='test.csv'): ···

    def handle_interrupt(self, signal, frame): ···

    def capture_packets(self): ···

    def train_capture_packets(self): ···


    def train_kNN(self): ···

    def test_kNN(self): ···

    def prediction(self): ···

    def publish_connected_devices(self): ···

    def get_local_network(self): ···

    def scan(self, ip_range):         ···

    def print_devices(self): ···
```

Fig No. 7.6.2 SYN_DoS_DDoS Class

This image depicts all the functions in a class required to detect SYN_DoS_DDoS attacks

7.6.3 Default Dataset

- Benign_Traffic.csv and Attack_Traffic.csv

7.6.4 Dataset Features

- 'Highest Layer'
- 'Transport Layer'
- 'Source IP’

- 'Dest IP'
- 'Source Port'
- 'Dest Port'
- 'Packet Length'
- ‘SYN’,’ACK’,’FIN’,’RST’ (Separate columns)
- 'Packets/Time'
- 'Time Between Packets'
- 'target'

7.6.5 Features Significance

The dataset contains critical characteristics for network traffic analysis, allowing for a thorough knowledge of packet behaviour.

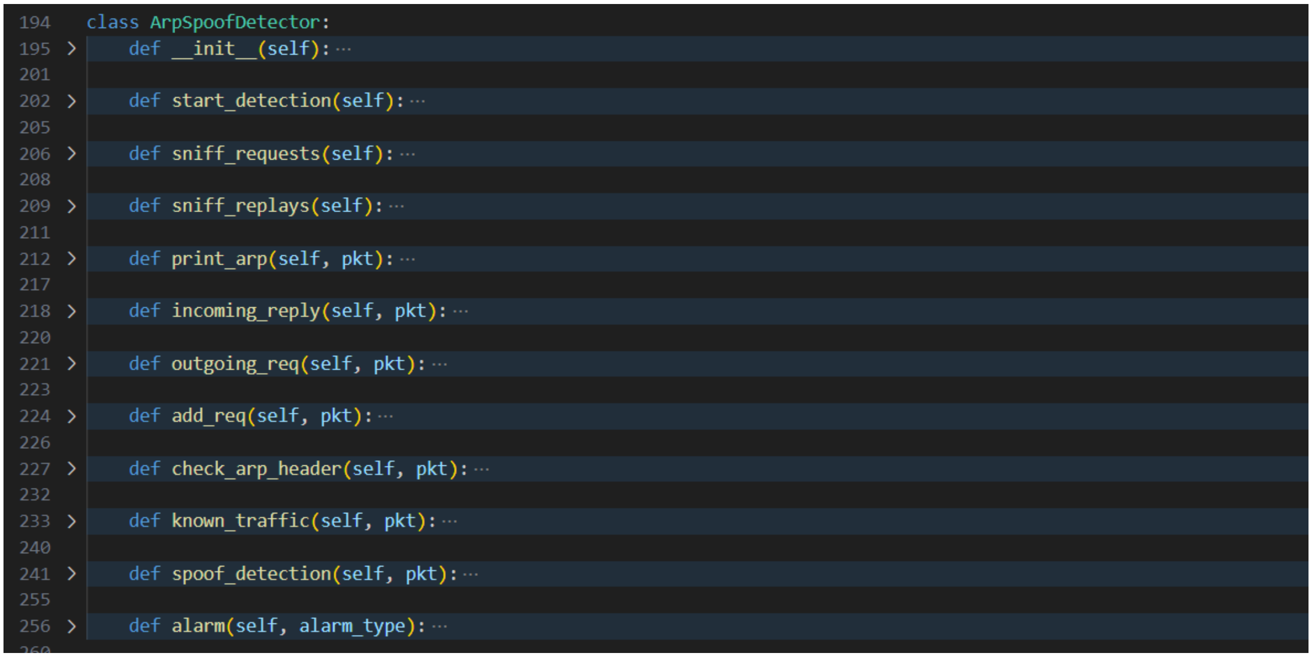
These include 'Highest Layer,' capturing the top protocol of the sniffed packet; 'Transport Layer,' indicating whether the packet is UDP or TCP; 'Source IP' and 'Dest IP' for source and destination IP addresses; 'Source Port' and 'Dest Port' for respective port information; 'Packet Length' denoting the size of each packet; and flags such as 'SYN,' 'ACK,' 'FIN,' and 'RST' indicating specific settings. Other elements include 'Packets/Time,' which represents the packet flow rate, and 'Time Between Packets,' which measures the time delay between consecutive packets. The 'target' field categorises entries as 'Attack' or 'Normal.'

7.7 ARP Spoofing Detection class

7.7.1 Algorithm

- Network traffic analysis, timestamps, and IP-MAC pair monitoring for anomaly detection.
- Relies on predefined rules and heuristics for effective ARP spoofing attack identification.

7.7.2 Interfaces



```
class ArpSpoofDetector:
    def __init__(self): ···

    def start_detection(self): ···

    def sniff_requests(self): ···

    def sniff_replays(self): ···

    def print_arp(self, pkt): ···

    def incoming_reply(self, pkt): ···

    def outgoing_req(self, pkt): ···

    def add_req(self, pkt): ···

    def check_arp_header(self, pkt): ···

    def known_traffic(self, pkt): ···

    def spoof_detection(self, pkt): ···

    def alarm(self, alarm_type): ···
```

Fig No. 7.7.2 ARP Spoofing Detector Class

This image depicts all the functions in a class required to detect ARP Spoofing attacks

7.7.3 Default Dataset

- Derived from real-time ARP traffic

7.7.4 Features Significance

The dataset utilized by the ArpSpoofDetector is derived from real-time ARP traffic, emphasizing ARP requests and replies.

These dataset parts, which include IP addresses, MAC addresses, ARP operation codes, and timestamps, are critical in detecting ARP spoofing attempts. Monitoring ARP requests and replies, the detector identifies inconsistencies, such as changes in IP-MAC pairs or unexpected alterations in network behavior.

Timestamps are pivotal for assessing the recency of ARP requests, aiding in the identification of potential spoofing attempts. The dynamic nature of the dataset allows the detector to adapt to evolving network conditions, bolstering its effectiveness in real-world scenarios.

7.8 Packet Filtering

7.8.1 Network Sniffer Implementation:

- Developed using Scapy for robust network security.
- Operates as a vigilant guardian by capturing and scrutinizing packets on a designated network interface.

7.8.2 In-depth Packet Analysis:

- Utilizes Scapy's versatile capabilities to extract essential protocol information from packet headers.
- Dynamically engages with specialized detector classes based on identified protocols.

7.8.3 Specialized Detector Classes:

- Crafted to predict and discern specific network threats or anomalies.
- Utilizes machine learning or rule-based approaches for ongoing evaluation of network traffic.

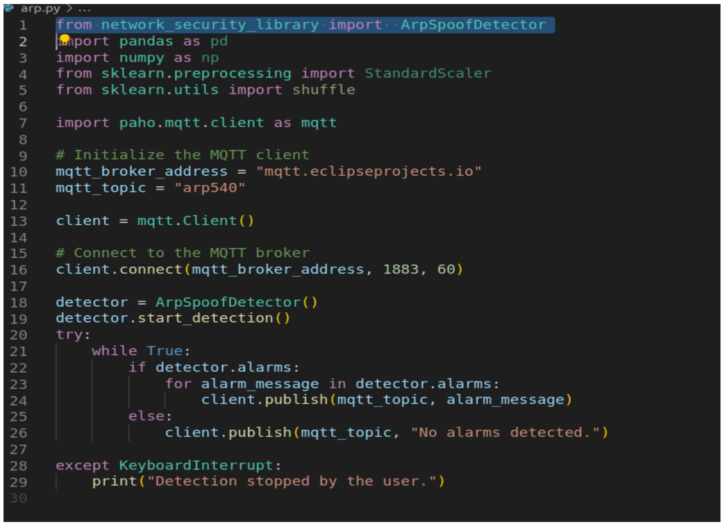
7.8.4 Real-time Adaptation:

- Facilitates swift adaptation to evolving network conditions.
- Enhances the system's ability to proactively respond to emerging threats through real-time interaction.

7.9 Example usage of the library

7.9.1 ArpSpoofDetector Usage

Fig 7.9.1 shows the usage of ArpSPoofDetector class and monitors the traffic and also publishes the results to MQTT broker.



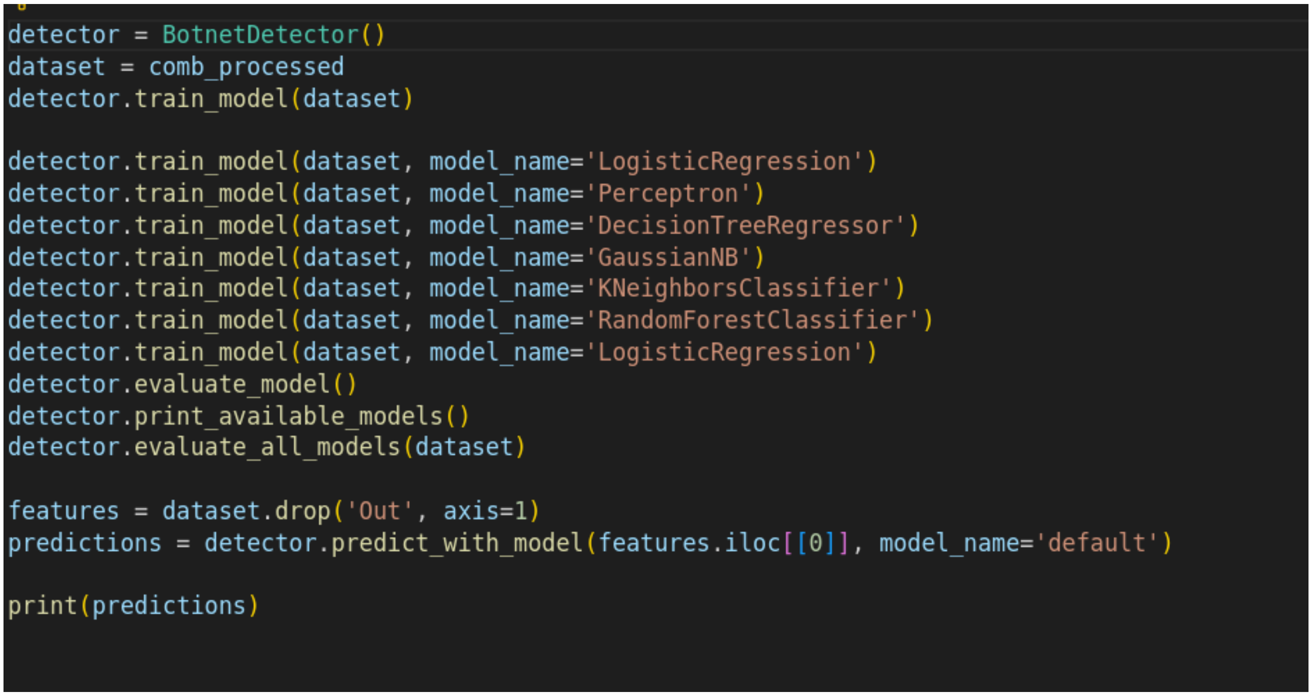
arp.py > ...

```python
from network_security_library import  ArpSpoofDetector
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle

import paho.mqtt.client as mqtt

# Initialize the MQTT client
mqtt_broker_address = "mqtt.eclipseprojects.io"
mqtt_topic = "arp540"

client = mqtt.Client()

# Connect to the MQTT broker
client.connect(mqtt_broker_address, 1883, 60)

detector = ArpSpoofDetector()
detector.start_detection()
try:
    while True:
        if detector.alarms:
            for alarm_message in detector.alarms:
                client.publish(mqtt_topic, alarm_message)
        else:
            client.publish(mqtt_topic, "No alarms detected.")

except KeyboardInterrupt:
    print("Detection stopped by the user.")
```

Fig 7.9.1 code using ArpSpoofDetector Clas

7.9.2 BotNetDetector Class

Figure 7.9.2 illustrates the training process of the existing models within the class, along with the subsequent evaluation and prediction of values for static data.

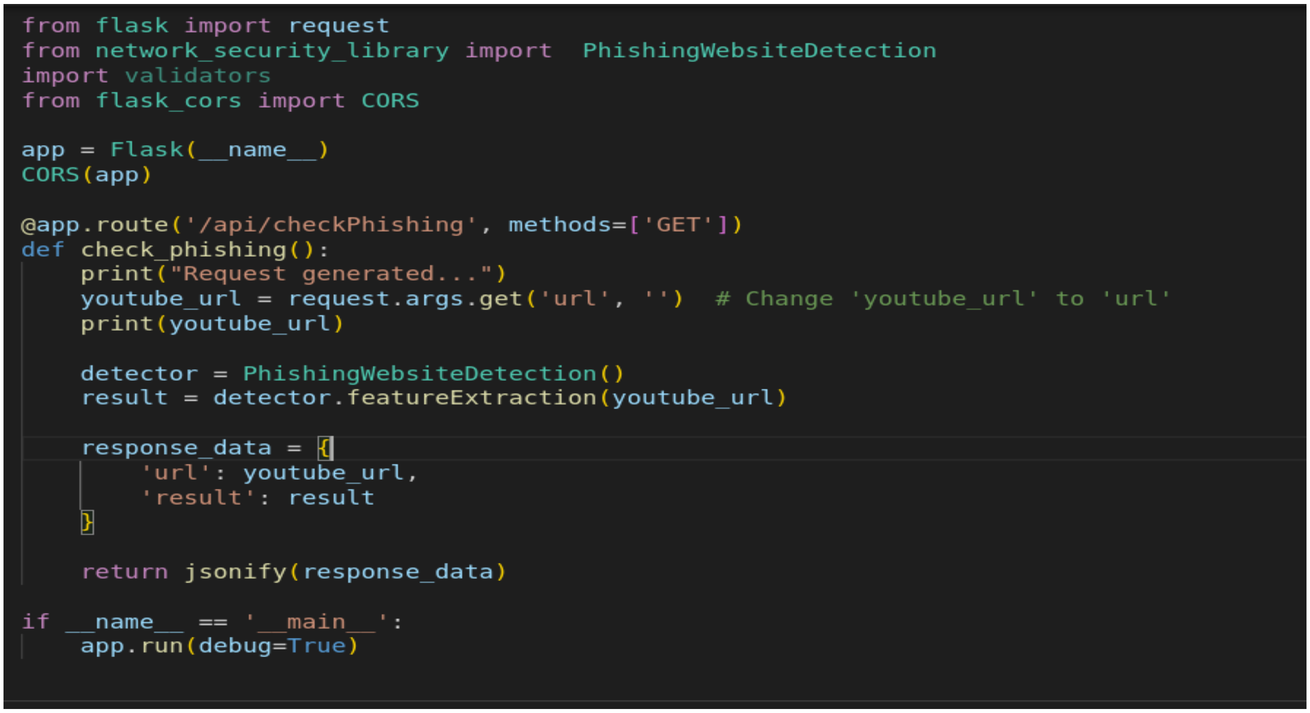


```
detector = BotnetDetector()
dataset = comb_processed
detector.train_model(dataset)

detector.train_model(dataset, model_name='LogisticRegression')
detector.train_model(dataset, model_name='Perceptron')
detector.train_model(dataset, model_name='DecisionTreeRegressor')
detector.train_model(dataset, model_name='GaussianNB')
detector.train_model(dataset, model_name='KNeighborsClassifier')
detector.train_model(dataset, model_name='RandomForestClassifier')
detector.train_model(dataset, model_name='LogisticRegression')
detector.evaluate_model()
detector.print_available_models()
detector.evaluate_all_models(dataset)

features = dataset.drop('Out', axis=1)
predictions = detector.predict_with_model(features.iloc[[0]], model_name='default')

print(predictions)
```

Fig 7.9.2 code using BotnetDetector Class

7.9.3 PhishingWebsiteDetection

Figure 7.9.3 illustrates This Python script using Flask to create a simple API endpoint (/api/checkPhishing) for checking if a provided URL is associated with phishing. It employs a custom module, PhishingWebsiteDetection, to perform phishing detection based on feature extraction. The extracted result is returned as a JSON response, including the original URL.

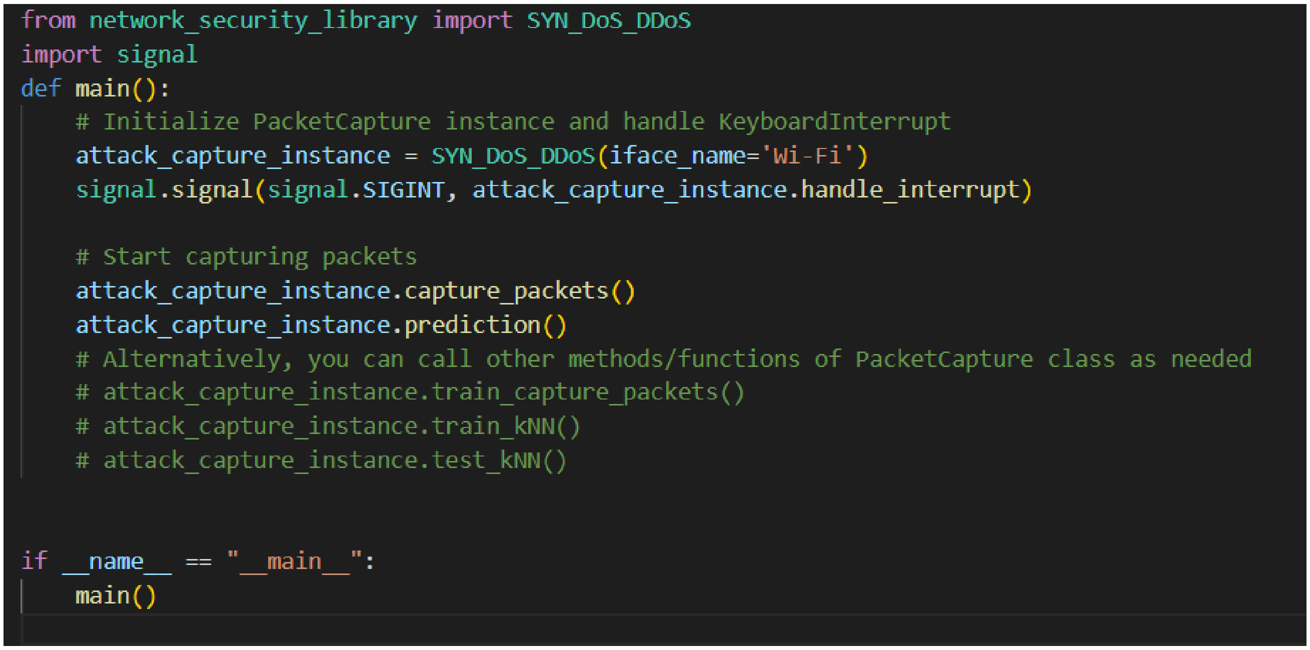


```
from flask import request
from network_security_library import  PhishingWebsiteDetection
import validators
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/api/checkPhishing', methods=['GET'])
def check_phishing():
    print("Request generated...")
    youtube_url = request.args.get('url', '')  # Change 'youtube_url' to 'url'
    print(youtube_url)

    detector = PhishingWebsiteDetection()
    result = detector.featureExtraction(youtube_url)

    response_data = {
        'url': youtube_url,
        'result': result
    }

    return jsonify(response_data)

if __name__ == '__main__':
    app.run(debug=True)
```

Fig 7.9.3 code using PhishingWebsiteDetection Class

7.9.4 SYN DoS and DDoS Detection

Figure 7.9.4 depicts how a user can import the class SYN_DoS_DDoS along with its functions in order to detect SYN Denial of Services attacks. The predictions are pushed onto an application dashboard where the traffic is classified in real-time.

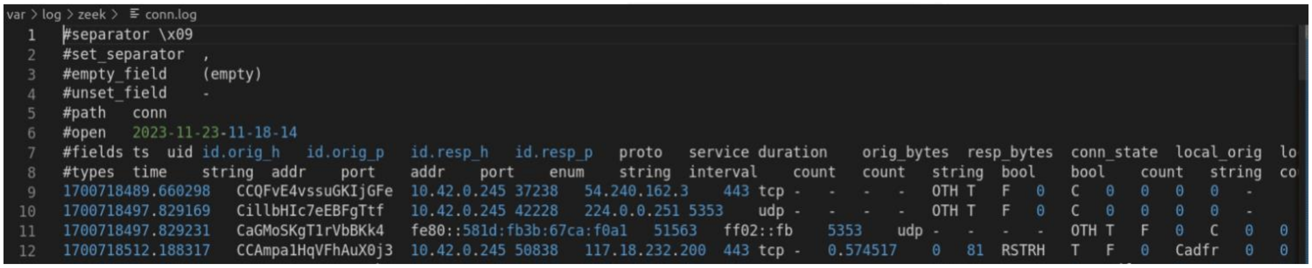


```python
from network_security_library import SYN_DoS_DDoS
import signal
def main():
    # Initialize PacketCapture instance and handle KeyboardInterrupt
    attack_capture_instance = SYN_DoS_DDoS(iface_name='Wi-Fi')
    signal.signal(signal.SIGINT, attack_capture_instance.handle_interrupt)

    # Start capturing packets
    attack_capture_instance.capture_packets()
    attack_capture_instance.prediction()
    # Alternatively, you can call other methods/functions of PacketCapture class as needed
    # attack_capture_instance.train_capture_packets()
    # attack_capture_instance.train_kNN()
    # attack_capture_instance.test_kNN()


if __name__ == "__main__":
    main()
```

Fig 7.9.4 Code using SYN_DoS_DDoS Class

CHAPTER - 8

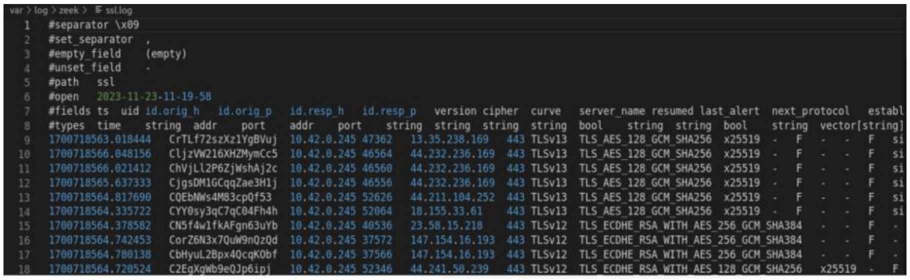
RESULTS AND DISCUSSION



```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   conn
#open   2023-11-23-11-18-14
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   proto   service duration    orig_bytes  resp_bytes  conn_state  local_orig  lo
#types  time    string  addr    port    addr    port    enum    string  interval    count   count   string  bool    bool    count   string  co
1700718489.660298   CCQFvE4vssuGKIjGFe  10.42.0.245 37238   54.240.162.3    443 tcp -   -   -   -   OTH T   F   0   C   0   0   0   0   -
1700718497.829169   CillbHIc7eEBFgTtf   10.42.0.245 42228   224.0.0.251 5353    udp -   -   -   -   OTH T   F   0   C   0   0   0   0   -
1700718497.829231   CaGMoSKgT1rVbBKk4   fe80::581d:fb3b:67ca:f0a1   51563   ff02::fb    5353    udp -   -   -   -   OTH T   F   0   C   0   0
1700718512.188317   CCAmpa1HqVFhAuX0j3  10.42.0.245 50838   117.18.232.200  443 tcp -   0.574517    0   81  RSTRH   T   F   0   Cadfr   0   0
```

Fig No. 8.1 Connection logs

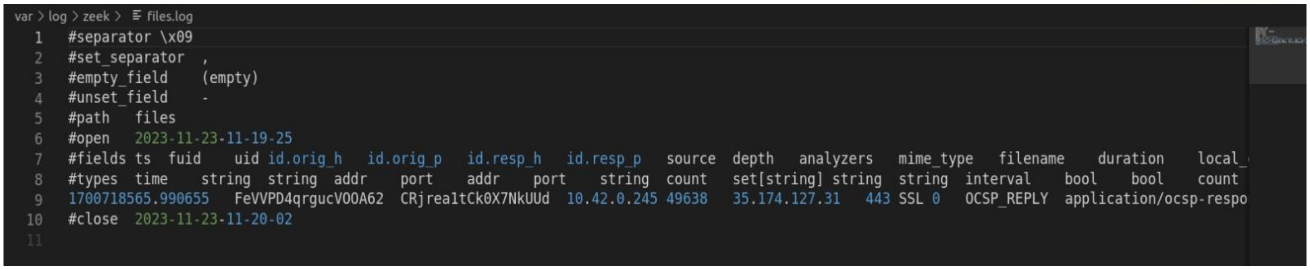
This image shows the file connection logs for a web server, which record the IP addresses and timestamps of all incoming and outgoing connections.



```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   ssl
#open   2023-11-23-11-19-58
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   version cipher  curve   server_name resumed last_alert  next_protocol   establ
#types  time    string  addr    port    addr    port    string  string  string  bool    string  string  bool    string  vector[string]
1700718563.018444   CrTLf72szXz1YgBVuj  10.42.0.245 47362   13.35.238.169   443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718566.048156   CljzVW216XHZMymCc5  10.42.0.245 46564   44.232.236.169  443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718566.021412   ChVjLl2P6ZjWshAj2c  10.42.0.245 46560   44.232.236.169  443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718565.637333   CjgsDM1GCqqZae3H1j  10.42.0.245 46556   44.232.236.169  443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718564.817690   CQEbNWs4M83cpQf53   10.42.0.245 52626   44.211.104.252  443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718564.335722   CYY0sy3qC7qC04Fh4h  10.42.0.245 52064   18.155.33.61    443 TLSv13  TLS_AES_128_GCM_SHA256  x25519  -   F   -   -   F   si
1700718564.378582   CN5f4w1fkAFgn63uYb  10.42.0.245 40536   23.58.15.218    443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384    -   -   F   -
1700718564.742453   CorZ6N3x7QuW9nQzQd  10.42.0.245 37572   147.154.16.193  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384    -   -   F   -
1700718564.780138   CbHyuL2Bpx4QcqKObf  10.42.0.245 37566   147.154.16.193  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384    -   -   F   -
1700718564.720524   C2EgXgWb9eQJp6ipj   10.42.0.245 52346   44.241.50.239   443 TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256    x25519  -   F
```

Fig No. 8.2 SSL logs(Secure Sockets Layer)

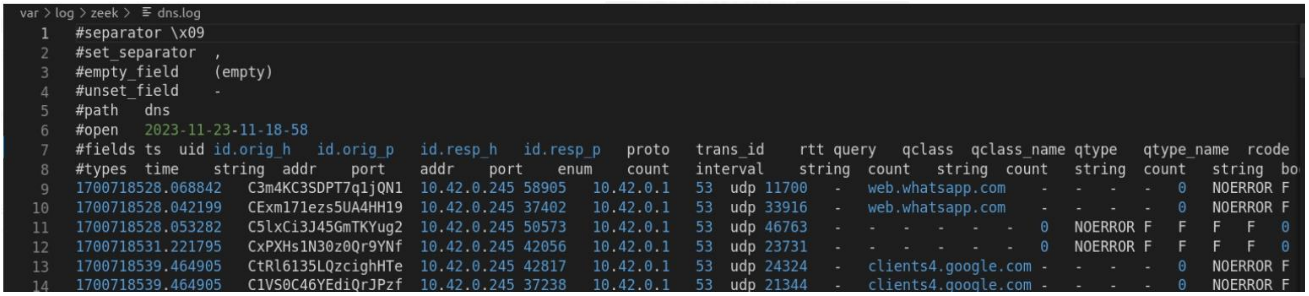
This image depicts the activity of SSL (Secure Sockets Layer) encryption protocols within a network, providing insights into secure communication channels and potential security vulnerabilities. The visual representation of SSL connection events helps network administrators identify abnormal patterns and potential threats to safeguard sensitive data.



```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   files
#open   2023-11-23-11-19-25
#fields ts  fuid    uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   source  depth   analyzers   mime_type   filename    duration    local_
#types  time    string  string  addr    port    addr    port    string  count   set[string] string  string  interval    bool    bool    count
1700718565.990655   FeVVPD4qrgucVOOA62  CRjrea1tCk0X7NkUUd  10.42.0.245 49638   35.174.127.31   443 SSL 0   OCSP_REPLY  application/ocsp-respo
#close  2023-11-23-11-20-02
```

Fig No. 8.3 Files log

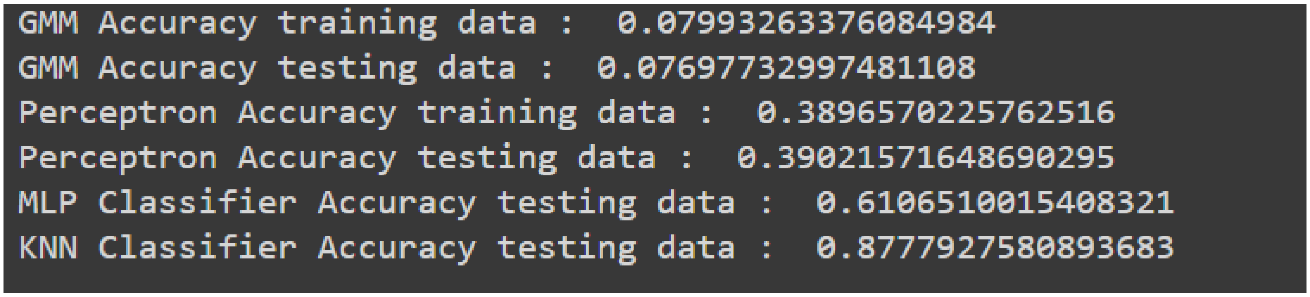
The image shows a log file from a web server, which contains information about incoming requests and responses.



```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   dns
#open   2023-11-23-11-18-58
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   proto   trans_id    rtt query   qclass  qclass_name qtype   qtype_name  rcode
#types  time    string  addr    port    addr    port    enum    count   interval    string  count   string  count   string  count   string  bo
1700718528.068842   C3m4KC3SDPT7q1jQN1  10.42.0.245 58905   10.42.0.1   53  udp 11700   -   web.whatsapp.com    -   -   -   -   0   NOERROR F
1700718528.042199   CExm171ezs5UA4HH19  10.42.0.245 37402   10.42.0.1   53  udp 33916   -   web.whatsapp.com    -   -   -   -   0   NOERROR F
1700718528.053282   C5lxCi3J45GmTKYug2  10.42.0.245 50573   10.42.0.1   53  udp 46763   -   -   -   -   -   -   0   NOERROR F   F   F   F   0
1700718531.221795   CxPXHs1N30z0Qr9YNf  10.42.0.245 42056   10.42.0.1   53  udp 23731   -   -   -   -   -   -   0   NOERROR F   F   F   F   0
1700718539.464905   CtRl6135LQzcighHTe  10.42.0.245 42817   10.42.0.1   53  udp 24324   -   clients4.google.com -   -   -   -   0   NOERROR F
1700718539.464905   C1VS0C46YEdiQrJPzf  10.42.0.245 37238   10.42.0.1   53  udp 21344   -   clients4.google.com -   -   -   -   0   NOERROR F
```

Fig No. 8.4 DNS logs(Domain Name Server)

The image shows a log of file connections on a server, including the IP addresses, ports, and protocols used.

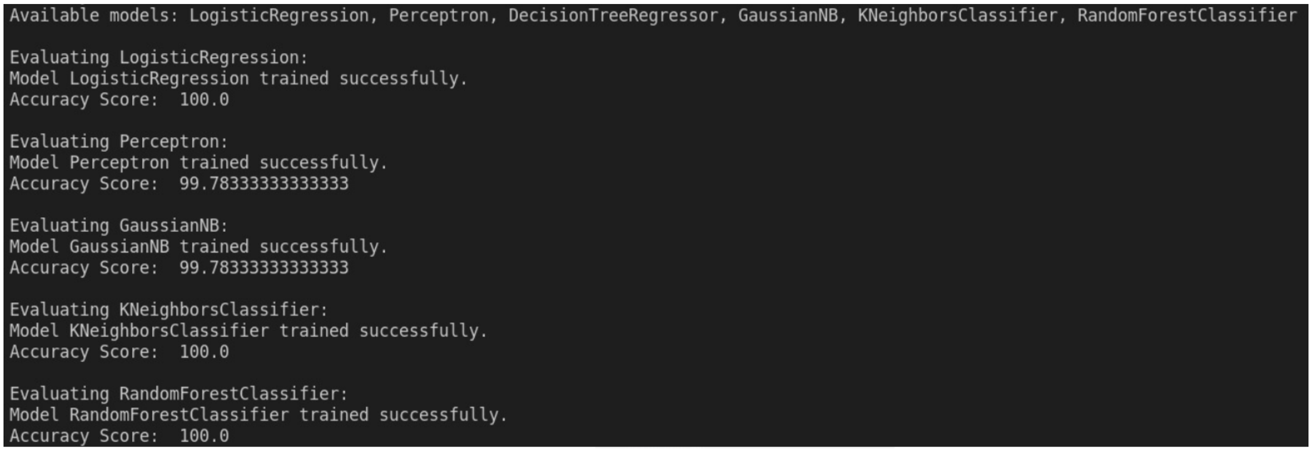


```
GMM Accuracy training data :  0.07993263376084984
GMM Accuracy testing data :  0.07697732997481108
Perceptron Accuracy training data :  0.3896570225762516
Perceptron Accuracy testing data :  0.39021571648690295
MLP Classifier Accuracy testing data :  0.6106510015408321
KNN Classifier Accuracy testing data :  0.8777927580893683
```

Fig No. 8.5 DDoS Detector Model accuracy

Botnet Detector

Logistic Regression is a powerful machine learning technique that detects botnet activity using binary categorization. Other models include the perceptron, Gaussian Naive Bayes, K-Neighbors Classifier, Random Forest Classifier, and Decision Tree Regressor. The CTU-13 dataset, a multivariate, sequential data collection, contains essential network analysis features. It gives temporal information on network connections, protocol and port data, network connection origin and destination, direction and state attributes, and service type indicators. Total bytes, total packets, and source bytes are quantitative measurements that quantify the amount of data transmitted. The dataset's significance stems from its capacity to handle scenarios with linear and clear feature connections, as well as capturing local patterns indicative of botnet behavior.

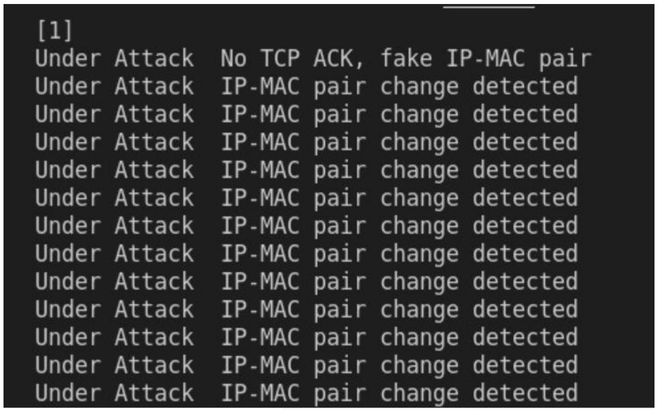


```
Available models: LogisticRegression, Perceptron, DecisionTreeRegressor, GaussianNB, KNeighborsClassifier, RandomForestClassifier

Evaluating LogisticRegression:
Model LogisticRegression trained successfully.
Accuracy Score:  100.0

Evaluating Perceptron:
Model Perceptron trained successfully.
Accuracy Score:  99.78333333333333

Evaluating GaussianNB:
Model GaussianNB trained successfully.
Accuracy Score:  99.78333333333333

Evaluating KNeighborsClassifier:
Model KNeighborsClassifier trained successfully.
Accuracy Score:  100.0

Evaluating RandomForestClassifier:
Model RandomForestClassifier trained successfully.
Accuracy Score:  100.0
```

Fig No. 8.6 BOTNET Detector Model Accuracy

ARP Spoofing Detection

To detect ARP spoofing attacks, the ArpSpoofDetector employs machine learning approaches. It detects anomalies through network traffic analysis, timestamps, and IP-MAC pair monitoring. The detector is based on real-time network traffic captured by the sniffer, allowing it to adapt to the specific network environment under observation. The collection contains IP addresses, MAC addresses, ARP operation codes, and timestamps, all of which are essential for detecting ARP spoofing attempts. The dataset's dynamic nature increases its usefulness in real-world circumstances.



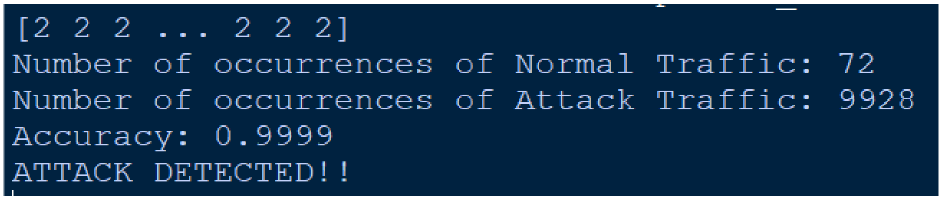
```
[1]
Under Attack  No TCP ACK, fake IP-MAC pair
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
Under Attack  IP-MAC pair change detected
```

Fig No. 8.7 ARP Spoofing

SYN DOS, DDoS Detector

The tool initially performs packet capture, which allows all packets to be captured for a specified period of time. Following that, it is determined if an IP address is IPv4 or IPv6. The situation in which the user pauses the packet capture is addressed by saving all packets up to that moment and sending them to the analysis phase in the Fog Device. All relevant data is then extracted, concatenated to the data frame, and saved to an Excel file. Two packet capture classes are used, Attack and Benign Traffic, and they are combined into a single dataset for analysis. The data is then preprocessed, with unnecessary columns removed, categorical characteristics label encoded, and the model trained using k-closest neighbours. To test the model, prevent overfitting, and provide the classification report, Stratified K-Fold Cross Validation is used.

There were two types of testing used: static testing and dynamic testing. Static testing involved running the model on attack traffic and counting the number of Syn flags. It was deemed an attack if the count of SYN flags with a value of 2 exceeded the count with a value of 1. In the case of dynamic testing, the k-nearest neighbours model was used, and any unnecessary columns in the test data were eliminated. The majority of SYN flags were examined, and if the majority was 2, the attack was classified as a SYN flood. Furthermore, the majority of IP addresses were examined, with a single IP address classified as a DoS attack and several IP addresses classified as a DDoS attack.

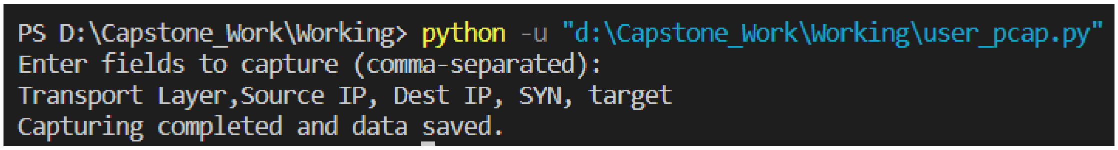


```
[2 2 2 ... 2 2 2]
Number of occurrences of Normal Traffic: 72
Number of occurrences of Attack Traffic: 9928
Accuracy: 0.9999
ATTACK DETECTED!!
```

Fig No. 8.8 Testing of the kNN model

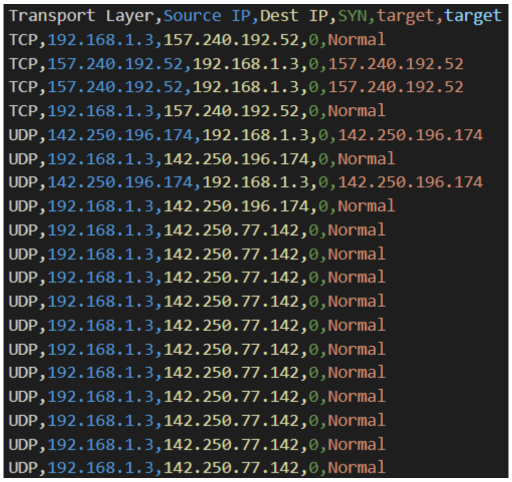
Packet-Capture instances were created for benign and attack traffic for the main function. The handle-Interrupt function was used to manage keyboard interruptions. First, Benign traffic was captured and before capturing attack traffic, a pause was added. After combining benign and attack data, the model was trained and evaluated.

User's are given the freedom to capture the field that they desire. As part of the library, a default list of features that may be captured have been provided. This can be as per the user's convenience.



```
PS D:\Capstone_Work\Working> python -u "d:\Capstone_Work\Working\user_pcap.py"
Enter fields to capture (comma-separated):
Transport Layer,Source IP, Dest IP, SYN, target
Capturing completed and data saved.
```

Fig No. 8.9 Custom features to be sniffed

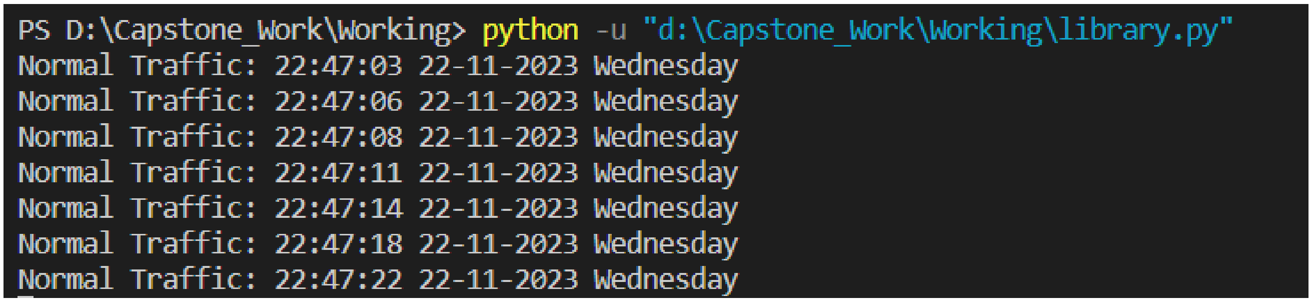


```
Transport Layer,Source IP,Dest IP,SYN,target,target
TCP,192.168.1.3,157.240.192.52,0,Normal
TCP,157.240.192.52,192.168.1.3,0,157.240.192.52
TCP,157.240.192.52,192.168.1.3,0,157.240.192.52
TCP,192.168.1.3,157.240.192.52,0,Normal
UDP,142.250.196.174,192.168.1.3,0,142.250.196.174
UDP,192.168.1.3,142.250.196.174,0,Normal
UDP,142.250.196.174,192.168.1.3,0,142.250.196.174
UDP,192.168.1.3,142.250.196.174,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
UDP,192.168.1.3,142.250.77.142,0,Normal
```

Fig No. 8.10 Custom features captured in CSV file

These are the data which are present in the csv files

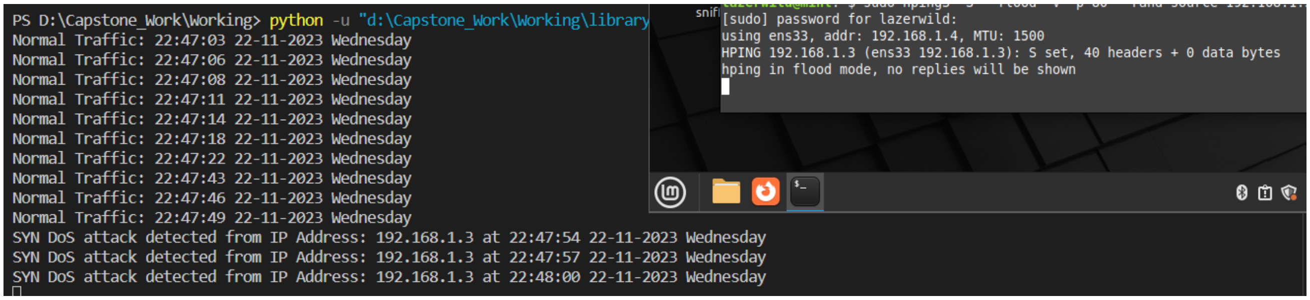
Once the Framework beings to run, it captures and labels necessary fields and stores them in a csv file and this data is encoded and provided to the model for prediction. The output may be as shown below.



```
PS D:\Capstone_Work\Working> python -u "d:\Capstone_Work\Working\library.py"
Normal Traffic: 22:47:03 22-11-2023 Wednesday
Normal Traffic: 22:47:06 22-11-2023 Wednesday
Normal Traffic: 22:47:08 22-11-2023 Wednesday
Normal Traffic: 22:47:11 22-11-2023 Wednesday
Normal Traffic: 22:47:14 22-11-2023 Wednesday
Normal Traffic: 22:47:18 22-11-2023 Wednesday
Normal Traffic: 22:47:22 22-11-2023 Wednesday
```

Fig No. 8.11 Normal Traffic

The image above shows the Normal traffic of packets i.e., No attack traffic.



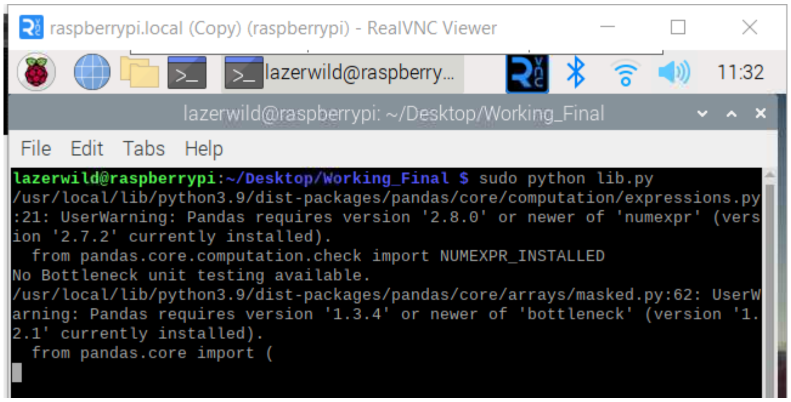
```
PS D:\Capstone_Work\Working> python -u "d:\Capstone_Work\Working\library
Normal Traffic: 22:47:03 22-11-2023 Wednesday
Normal Traffic: 22:47:06 22-11-2023 Wednesday
Normal Traffic: 22:47:08 22-11-2023 Wednesday
Normal Traffic: 22:47:11 22-11-2023 Wednesday
Normal Traffic: 22:47:14 22-11-2023 Wednesday
Normal Traffic: 22:47:18 22-11-2023 Wednesday
Normal Traffic: 22:47:22 22-11-2023 Wednesday
Normal Traffic: 22:47:43 22-11-2023 Wednesday
Normal Traffic: 22:47:46 22-11-2023 Wednesday
Normal Traffic: 22:47:49 22-11-2023 Wednesday
SYN DoS attack detected from IP Address: 192.168.1.3 at 22:47:54 22-11-2023 Wednesday
SYN DoS attack detected from IP Address: 192.168.1.3 at 22:47:57 22-11-2023 Wednesday
SYN DoS attack detected from IP Address: 192.168.1.3 at 22:48:00 22-11-2023 Wednesday
```

```
[sudo] password for lazerwild:
using ens33, addr: 192.168.1.4, MTU: 1500
HPING 192.168.1.3 (ens33 192.168.1.3): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Fig No. 8.12 Differentiates from Normal and Attack Traffic

The image above shows how Attack and Normal traffic are differentiated and we used hping3 to attack the target system. The attack has been launched by a tool, Hping3.

The below image displays how this framework may run on a Raspberry Pi

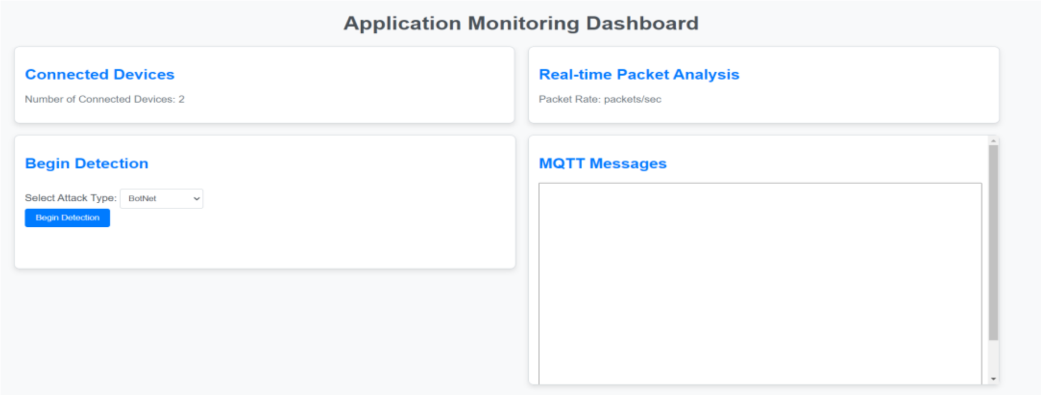

raspberrypi.local (Copy) (raspberrypi) - RealVNC Viewer
lazerwild@raspberry...
11:32
lazerwild@raspberrypi: ~/Desktop/Working_Final
File Edit Tabs Help

```
lazerwild@raspberrypi:~/Desktop/Working_Final $ sudo python lib.py
/usr/local/lib/python3.9/dist-packages/pandas/core/computation/expressions.py
:21: UserWarning: Pandas requires version '2.8.0' or newer of 'numexpr' (vers
ion '2.7.2' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
No Bottleneck unit testing available.
/usr/local/lib/python3.9/dist-packages/pandas/core/arrays/masked.py:62: UserW
arning: Pandas requires version '1.3.4' or newer of 'bottleneck' (version '1.
2.1' currently installed).
  from pandas.core import (
```

Fig No. 8.13 Running the code on Raspberry Pi

The above method within the Framework also provides a functionality to publish these predictions onto any application that the user desires, via MQTT. This method defines a MQTT publisher and the predicted data is sent to the MQTT server under the topic name, ‘packet_capture_results’. The user may subscribe to this topic name for viewing the results.

A Dashboard was built that behaves as a dashboard for an IoT network. It displays the results of the prediction in real-time as well as, mentioning how many devices are connected to the network.


Application Monitoring Dashboard

Connected Devices

Number of Connected Devices: 2

Real-time Packet Analysis

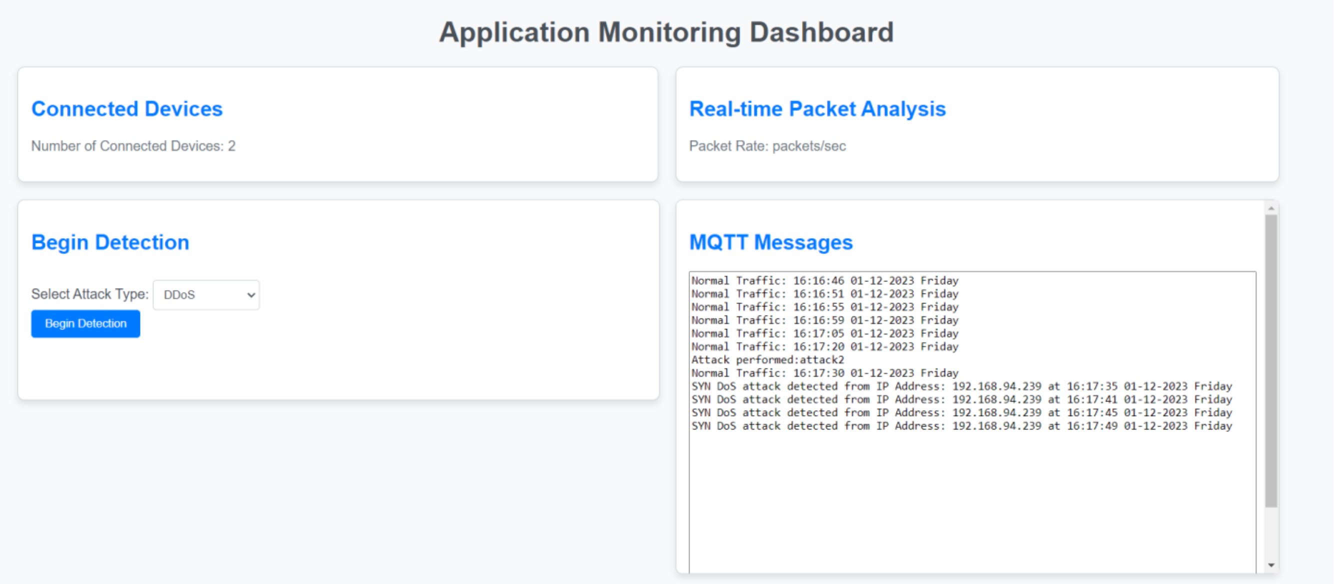
Packet Rate: packets/sec

Begin Detection

Select Attack Type: BotNet

MQTT Messages

Fig No. 8.14 Example of a Dashboard



Application Monitoring Dashboard

Connected Devices

Number of Connected Devices: 2

Real-time Packet Analysis

Packet Rate: packets/sec

Begin Detection

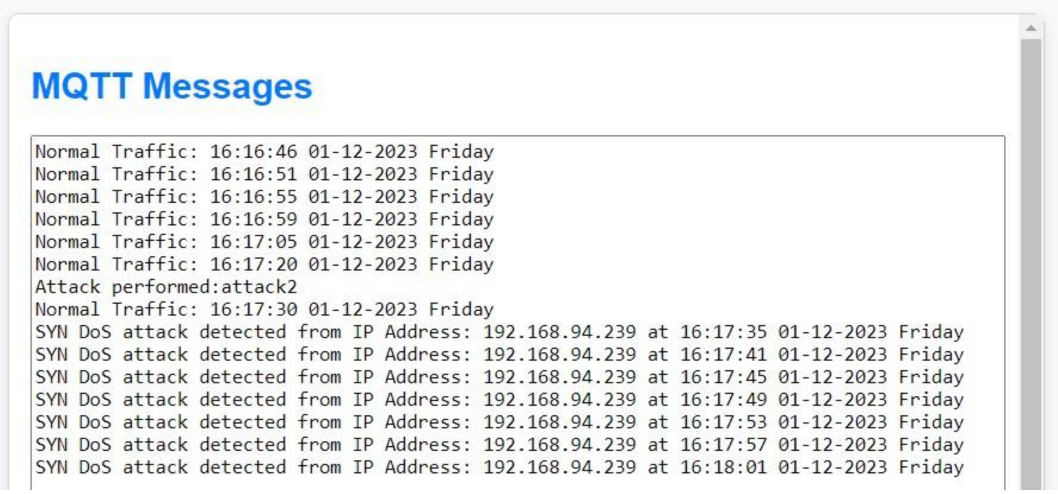
Select Attack Type: DDoS

MQTT Messages

```
Normal Traffic: 16:16:46 01-12-2023 Friday
Normal Traffic: 16:16:51 01-12-2023 Friday
Normal Traffic: 16:16:55 01-12-2023 Friday
Normal Traffic: 16:16:59 01-12-2023 Friday
Normal Traffic: 16:17:05 01-12-2023 Friday
Normal Traffic: 16:17:20 01-12-2023 Friday
Attack performed:attack2
Normal Traffic: 16:17:30 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:35 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:41 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:45 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:49 01-12-2023 Friday
```

Fig No. 8.15 Predictions pushed onto dashboard using MQTT.

The predictions that the ML Model made is pushed to the dashboard using MQTT and the Attack and Normal traffic is being differentiated.



MQTT Messages

```
Normal Traffic: 16:16:46 01-12-2023 Friday
Normal Traffic: 16:16:51 01-12-2023 Friday
Normal Traffic: 16:16:55 01-12-2023 Friday
Normal Traffic: 16:16:59 01-12-2023 Friday
Normal Traffic: 16:17:05 01-12-2023 Friday
Normal Traffic: 16:17:20 01-12-2023 Friday
Attack performed:attack2
Normal Traffic: 16:17:30 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:35 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:41 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:45 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:49 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:53 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:17:57 01-12-2023 Friday
SYN DoS attack detected from IP Address: 192.168.94.239 at 16:18:01 01-12-2023 Friday
```

Fig No. 8.16 A closer look at the predictions

CHAPTER - 9

CONCLUSION AND FUTURE WORK

In conclusion, the proposed cybersecurity framework emerges as a robust solution for bolstering the security infrastructure of IoT networks. The strategic use of a Raspberry Pi 4 as a fog gateway reflects an innovative approach to detect early cyber threats, encompassing a range of attacks such as phishing, botnet attack, SYN DoS and DDoS, and ARP spoofing. The framework's versatility is evident in its diverse applications that have been shown in the Results section of the report, including a Chrome extension for phishing detection and a real-time ARP spoofing analysis displayed on an interactive dashboard.

This adaptability extends to the framework's core, allowing users to customize their security protocols by selecting attack models, datasets, and testing scenarios. The amalgamation of these powerful detection mechanisms with practical, real-world implementations underscores its significance in addressing the dynamic challenges of cybersecurity.

The envisioned future work for the cybersecurity framework encompasses several key dimensions, each tailored to elevate the system's efficacy and adaptability. Firstly, an emphasis on Enhanced Model Diversity is pivotal, involving the incorporation of a broader spectrum of machine learning models. Dynamic Dataset Integration stands as another critical frontier, with a focus on implementing mechanisms for real-time dataset updates. This proactive approach ensures that the system remains at the forefront of cybersecurity by promptly adapting to emerging threats, thereby sustaining its effectiveness over time.

Lastly, a proactive stance towards Community Contributions is integral to the framework's evolution. Encouraging and facilitating community engagement involves the development of comprehensive documentation, tutorials, and guidelines. This initiative aims to empower users and developers to extend the framework seamlessly, introducing new attack classes and functionalities to collectively fortify the cybersecurity landscape.

REFERENCES

1.M. Satyanarayanan, ‘‘The emergence of edge computing,’’ Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017.

2.W. S. Shi, X. Z. Zhang, and Y. F. Wang, ‘‘Edge computing: State-of-the-art and future directions,’’ J. Comput. Res. Develop., vol. 56, no. 1, pp. 1–21, 2019.

3.W. Shi, H. Sun, J. Cao, Q. Zhang, and W. Liu, ‘‘Edge computing-an emerging computing model for the Internet of everything era,’’ J. Comput. Res. Develop., vol. 54, no. 5, pp. 907–924, May 2017.

4. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, ‘‘Edge computing: Vision and challenges,’’ IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

5.K. Angrishi, “Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets,” arXiv preprint arXiv:1702.03681, 2017.

6.S. Naaz, “Detection of phishing in internet of things using machine learning approach,” International Journal of Digital Crime and Forensics (IJDCF), vol. 13, no. 2, pp. 1–15, 2021.

7.L. Toms, “Common cyber attacks in the iot-threat alert on a grand scale,” 2018.

8.A. K. Pathak, S. Saguna, K. Mitra, and C. ˚Ahlund, “Anomaly detection using machine learning to discover sensor tampering in iot systems,” in ICC 2021 - IEEE International Conference on Communications, pp. 1–6, 2021.

9.A. Dutta and S. Kant, “Implementation of cyber threat intelligence platform on internet of things (iot) using tinyml approach for deceiving cyber invasion,” in 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), pp. 1–6, 2021.

10. S. O. Uwagbole, W. J. Buchanan, and L. Fan, “Applied machine learning predictive analytics to sql injection attack detection and prevention,” in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1087–1090, 2017.

11. S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, “Proposed embedded security framework for internet of things (iot),” in 2011 2nd International Conference on Wireless Communication, Vehicular

Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), pp. 1–5, 2011.

12. Q. Chen, H. Chen, Y. Cai, Y. Zhang, and X. Huang, "Denial of service attack on iot system," in 2018 9th International Conference on Information Technology in Medicine and Education (ITME), pp. 755–758, 2018.

13. Munshi, N. A. Alqarni, and N. Abdullah Almalki, "Ddos attack on iot devices," in 2020 3rd International Conference on Computer Applications Information Security (ICCAIS), pp. 1–5, 2020.

14. H. Abdulla, H. Al-Raweshidy, and W. S Awad, "Arp spoofing detection for iot networks using neural networks," in Proceedings of the Industrial Revolution & Business Management: 11th Annual PwR Doctoral Symposium (PWRDS), 2020.

15. Y. Shah and S. Sengupta, "A survey on classification of cyber-attacks on iot and iiot devices," in 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pp. 0406–0413, IEEE, 2020.

16. A. Rajan, J. Jithish, and S. Sankaran, "Sybil attack in iot: Modelling and defenses," in 2017 International Conference on Advances in Computing,Communications and Informatics (ICACCI), pp. 2323–2327, 2017.

17. G. Rajendran, R. S. Ragul Nivash, P. P. Parthy, and S. Balamurugan, "Modern security threats in the internet of things (iot): Attacks and countermeasures," in 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–6, 2019.

18. E. M. de Lima Pinto, R. Lachowski, M. E. Pellenz, M. C. Penna, and R. D. Souza, "A machine learning approach for detecting spoofing attacks in wireless sensor networks," in 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), pp. 752–758, IEEE, 2018.

19. T. Gaber, A. El-Ghamry, and A. E. Hassanien, "Injection attack detection using machine learning for smart iot applications," Physical Communication, vol. 52, p. 101685, 2022.

20. M. Burhan, R. A. Rehman, B. Khan, and B.-S. Kim, "Iot elements, layered architectures and security issues: A comprehensive survey," sensors, vol. 18, no. 9, p. 2796, 2018.

21. K. Aarika, M. Bouhlal, R. A. Abdelouahid, S. Elfilali, and E. Benlahmar, “Perception layer security in the internet of things,” Procedia Computer Science, vol. 175, pp. 591–596, 2020.

22. I.-G. Lee, K. Go, and J. H. Lee, “Battery draining attack and defense against power saving wireless lan devices,” Sensors, vol. 20, no. 7, p. 2043, 2020.

23. S. Pirbhulal, N. Pombo, V. Felizardo, N. Garcia, A. H. Sodhro and S. C. Mukhopadhyay, "Towards Machine Learning Enabled Security Framework for IoT-based Healthcare," *2019 13th International Conference on Sensing Technology (ICST)*, Sydney, NSW, Australia, 2019, pp. 1-6, doi: 10.1109/ICST46873.2019.9047745.
24. M. Nobakht, V. Sivaraman and R. Boreli, "A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow," *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Salzburg, Austria, 2016, pp. 147-156, doi: 10.1109/ARES.2016.64.

25. J. Pacheco and S. Hariri, "IoT Security Framework for Smart Cyber Infrastructures," *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Augsburg, Germany, 2016, pp. 242-247, doi: 10.1109/FAS-W.2016.58.

26. Karmous, Nader, et al. ”IoT real-time attacks classification framework using machine learning.” 2022 IEEE Ninth International Conference on Communications and Networking (ComNet). IEEE, 2022.

27. M. Bagaa, T. Taleb, J. B. Bernabe and A. Skarmeta, ”A Machine Learning Security Framework for Iot Systems,” in IEEE Access, vol. 8, pp. 114066-114077, 2020, doi: 10.1109/ACCESS.2020.2996214.

28. Donalek, Ciro. ”Supervised and unsupervised learning.” Astronomy Colloquia. USA. Vol. 27. 2011.

29. Delplace, Antoine, Sheryl Hermoso, and Kristofer Anandita. ”Cyber attack detection thanks to machine learning algorithms.” arXiv preprint arXiv:2001.06309 (2020).

30. So, Kenn. ”Chapter 13: Multi-layer Perceptrons (MLPs)” Kenn So Github.

31. Srivastava, Tavish. ”A Complete Guide to K-Nearest Neighbors (Updated 2023)” AnalyticsVidhya, 20th October 2023 https://www.analyticsvidhya.com/blog/2018/03/introduction-kneighbours-algorithm-clustering/

32. Zhai, Junhai, et al. ”Autoencoder and its various variants.” 2018 IEEE international conference on systems, man, and cybernetics (SMC). IEEE, 2018.

33. M. Kumar, M. Hanumanthappa and T. V. S. Kumar, "Intrusion Detection System using decision tree algorithm," *2012 IEEE 14th International Conference on Communication Technology*, Chengdu, China, 2012, pp. 629-634, doi: 10.1109/ICCT.2012.6511281.

34. R. Doshi, N. Apthorpe and N. Feamster, ”Machine Learning DDoS Detection for Consumer Internet of Things Devices,” 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 2018, pp. 29-35, doi: 10.1109/SPW.2018.00013.

35. M. S. Christo, J. J. Menandas, M. George and S. V. Nuna, "DDoS Detection using Multilayer Perceptron," 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2023, pp. 688-693, doi: 10.1109/ICESC57686.2023.10193406.
36. Q. A. Al-Haija, E. Saleh and M. Alnabhan, "Detecting Port Scan Attacks Using Logistic Regression," 2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT), Alkhobar, Saudi Arabia, 2021, pp. 1-5, doi: 10.1109/ISAECT53699.2021.9668562.
37. Assegie, Tsehay Admassu. "K-nearest neighbor based URL identification model for phishing attack detection." Indian Journal of Artificial Intelligence and Neural Networking 1 (2021): 18-21.
38. Rawat, Romil, and Shailendra Kumar Shrivastav. "SQL injection attack Detection using SVM." International Journal of Computer Applications 42.13 (2012): 1-4.