Applied Information Technology And Computer Science Vol. 4 No. 2 (2023) 139-158

AITCS

Homepage: http://publisher.uthm.edu.my/periodicals/index.php/aitcs
e-ISSN :2773-5141

Distributed Denial of Service (DDoS) Detection Tool Using Artificial Neural Network (ANN) for Homelab

Pathmahn a/l Murali[1], **Nur Ziadah Harun**[1]*

[1]Fakulti Sains Komputer dan Teknologi Maklumat,
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

DOI: https://doi.org/10.30880/aitcs.2023.04.02.009
Received 08 September 2023; Accepted 06 November 2023; Available online 30 November 2023

Abstract: A homelab is a personal server or cluster of servers kept in a residence and used for various purposes such as testing, development, or everyday use. These labs enable tech enthusiasts to create their own video streaming services, email hosting platforms, cloud storage, and other services without relying on large tech corporations. However, the increasing presence of advanced botnets and the search for vulnerable databases to steal and sell data from has posed a threat to many homelab users who do not have the budget or resources to implement a comprehensive Intrusion Detection System (IDS). To address these challenges, a proposal has been made to create a Distributed Denial of Service (DDoS) detection tool using Artificial Neural Networks (ANN) for homelab users. The goals of this proposed system are to design and develop an ANN-based DDoS detection tool for homelab users, and to test its effectiveness against real-life threats. The expected outcome is the implementation of an ANN capable of accurately detecting DDoS attacks with minimal false positives without hindering the performance of the server. Currently, the system currently has a recall and F1 score of 0.97 which indicates that the system can detect DDoS attack with fewer false positives.

Keywords: Artificial Intelligence, Artificial Neural Networks, Distributed Denial of Service, Homelabs, Intrusion Detection System

1. Introduction

In recent years, the high costs of cloud-based solutions and video subscription services have led to the emergence of a new community of IT enthusiasts consisting of developers, software engineers, and network engineers who have started building their own home labs. A homelab is a personal server or cluster of servers kept in a residence and used for various purposes such as testing, development, or everyday use [1]. The availability of used servers and enterprise-grade equipment has also contributed to the popularity of homelabs. However, the increasing presence of advanced botnets that scan the internet for vulnerable targets to be used as "zombies" and the search for vulnerable databases to steal and sell data from has posed a threat to many homelab users who do not have the budget or resources

*Corresponding author: nurziadah@uthm.edu.my

to implement a comprehensive intrusion detection system (IDS) to detect these threats. The lack of an affordable, lightweight solution that is easy to set up and use has made it difficult for homelab users to protect their systems. Most existing solutions require complex configurations and setup, and many do not utilize artificial intelligence to facilitate faster detection with fewer errors.

To address these challenges, the project has been made to create a distributed denial of service (DDoS) detection tool using artificial neural networks (ANN) for homelab users. The tool will be designed to work on low to mid-range server appliances, tested on a Raspberry Pi, and tested against simulated attacks. The target audience is developers and homelab owners, and the development process will use Agile method. The tool will detect attacks with high accuracy and minimal overhead, ensuring optimal server performance. Additionally, it features a user-friendly interface with push notifications for timely alerts

In this documentation, Section 2 discusses the related works and system that are currently available in the market and provides a comparison between the available system and the proposed system. Next, Section 3 will explain on methodologies and frameworks used for the development of this system. Section 4 will investigate the results and discussions of the proposed system. Finally, Section 5 concludes this documentation and looks ahead to future developments, including any subsequent documentation.

2. Literature Review

This section discusses the intrusion detection system, homelab, Distributed denial of service, Artificial Neural Networks (ANN) in Cyber Security

2.1 Intrusion Detection System (IDS)

IDS is a system or a device that is dedicated to monitoring and detecting anomalous and dangerous activities in a network [2]. An Intrusion Detection System works by constantly monitoring abnormal activities and behaviors' that trigger the rules set by the administrators. Intrusion Detection System has two methodologies that is used to detect anomalies in the network which are Signature-based and Anomaly-based detection [2] . A Signature-based IDS uses a database and a collection of attack signatures to detect threats in a network. An Anomaly-based IDS uses a machine learning model to alert administrators of an incursion. These two detection methodologies have their own drawbacks and advantages, but when employed in unison there can complement each other flaws [2] .

2.3 Homelab

Homelab is a label given to a server or a computer that resides locally in a home and used to host several services, applications, and virtual machines for development, testing and functional usage. These servers can be a simple desktop computer or a single board computer. Additionally, homelabs can be a cost-effective solution for hosting services and applications due to users using their own hardware and avoiding hosting service payment [1]. This can be especially beneficial for small businesses and individuals who may not have the budget for expensive hosting solutions. Overall, homelabs provide users with a high level of control, flexibility, and cost savings, making them an attractive option for tech enthusiasts and developers looking to host their own services and applications.

2.4 Distributed denial of service (DDoS)

Distributed denial of service (DDoS) attacks is a type of cyber-attack that flood a target network or system with traffic, rendering it unavailable to legitimate users [3]. These attacks have become a growing problem due to the increasing availability of cheap and powerful computing resources, the proliferation of insecure IoT devices, and the availability of DDoS-as-a-service offerings on the dark web [3]. Organizations use a variety of techniques to deter DDoS attacks, but they remain a significant problem and continue to evolve in terms of both the tactics used by attackers and the defenses deployed

by organizations. Continued research and development are needed to develop more effective ways of defending against these attacks and to better understand the motivations and tactics of attackers.

2.5 Artificial Neural Networks (ANN) in Cyber Security

Artificial Neural Networks (ANN) are known as artificial intelligence algorithm that is modelled after the operation of neurons in the human brain's neural system [4]. ANN are a popular algorithm amongst the artificial intelligence community because of its ability to solve complex problem which are unsolvable by other algorithms. The advancement of adversaries' tactics and the emergence of state sponsored APT's in these years have rendered standard security software useless towards cybersecurity attacks [4]. To overcome these problems, cybersecurity companies and providers have adopted artificial intelligence to enhance their products ability to detect incoming and persisting threats in an infrastructure. Traditional malware detection solutions depend on signature-based detection which is infective to deal with newer malware and advanced threats that infect the system [5]. Using ANN for malware detection helps in detecting more advanced malware and threats without needing to rely on signatures and common attack patterns by detecting anomalies in system activities and common attack signature.

2.6 Kafka

Apache Kafka is an open-source distributed streaming platform that can handle trillions of events, making it a highly scalable and fault-tolerant solution for processing real-time data [6]. Kafka's architecture is designed to prevent data loss by persisting messages on disk. It was initially developed at LinkedIn and later became an open-source project under the Apache Software Foundation. Kafka's architecture consists of several components, including producers, brokers, consumers, and consumer groups [6]. In conclusion, Apache Kafka is a powerful and versatile distributed streaming platform that enables the processing of massive volumes of real-time data

2.7 Streamlit

Streamlit is an open-source Python library that allows developers to easily create and share interactive web applications for machine learning and data science [7]. Its primary goal is to provide a fast, iterative, and interactive development experience, enabling developers to go from a Jupyter Notebook to a polished web application in just a matter of hours. Developers can write code and see the app updated instantly in the browser. Streamlit automatically detects changes in the code and offers to re-run the app, eliminating the need for manual build processes. Streamlit supports a wide range of interactive widgets, dataframes, tables, images, Markdown, plot rendering, maps, and textual content, making it a versatile tool for creating data-driven web applications [7].

2.8 Existing Intrusion Detection and Prevention System

We discuss Snort and Suricata as similar intrusion detection and prevention systems.

2.8.1 Snort

Snort is an open-source network intrusion detection and prevention system that was created in 1998 by Martin Roesch [8]. It uses a rule-based language to detect malicious activity in real time by analyzing network traffic and logging packets. The system can be used on Linux/Unix and Windows operating systems, and employs the "*libpcap*" and "*winpcap*" libraries, respectively. Snort can be configured to operate in three modes: sniffer, packet logger, and Network Intrusion Prevention Detection System (NIPDS) [8]. Sniffer mode scans incoming packets and logs them in a designated directory, while packet logger mode performs similar tasks but also saves the packets for forensic analysis. It is powered by the Snort rule-based language and the Misuses Detection Engine BASE. Snort can perform network traffic monitoring, anomaly detection, packet sniffing, and alert generation to ensure the security of the host network.

2.9 Suricata

Suricata is an open-source Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) developed by the Open Information Security Foundation (OISF) [9]. It was introduced in December 2009 and has since become widely deployed around the world. Like Snort, it is deployable on various platforms, including Windows, Linux, Mac, Unix, and FreeBSD, and supports scripting using the Lua language [10]. Suricata also can extract files from network traffic for analysis and has a built-in scripting engine called "*LuaJIT*".

2.10 Comparison of Existing Application with The Proposed Application

There is a wide array of existing options available for homelab intrusion detection systems. Among the chosen systems for analysis and comparison are Snort and Suricata, both of which share similarities with the proposed application.

Table 1: Comparison of Existing Application with DDoS Detection Tool Using ANN For Homelab

Application / Features	Snort	Suricata	Proposed System
Ease of Use	Yes	Yes	Yes
GUI	No (Requires third party application)	Yes	Yes
Artificial Intelligence	No	No	Yes (ANN)
Multithreading	No	Yes	No
Hardware Acceleration	Yes	Yes	No
Rules	Yes	Yes	No
Ease of Deployment	No	No	Yes
Containerization	No	No	Yes

The system is significantly better than the existing system because of its capabilities. The system contains a GUI whereas Snort requires a third-party application for GUI support. This makes it easier to use for some users. This make is easy to detect attacks with the visualization capabilities. The proposed system uses artificial intelligence in the form of an artificial neural network, whereas Snort and Suricata do not. This could potentially make it more effective at detecting and preventing attacks. Moreover, it has easy deployment methods with containerization and a lightweight infrastructure.

3. Methodology

3.2 Software requirements for developing the proposed system

In the development the system, it is essential to consider the software requirements needed to aid in the development process. The development of the DDoS Detection Tool Using ANN For Homelab requires specialized software. The specifications for the software used to create this system are laid forth in Table 2.

Table 2: Software Requirement

No	Software	Description
1.	Operating System	Microsoft Windows 10 Pro Build 19045
2.	Windows Subsystem for Linux	Version 1.2.5.0
3.	Linux Kernel	Version 5.15.79.1
4.	Containerization Software	Docker
5.	Development Container	TensorFlow and NVIDIA Container Toolkit
6.	Integrated Development Environment	JupyterLab and VS Code
7.	Web Server and Framework	Streamlit
8.	Database Engine	SQLite
9.	Traffic Capture and Analyzer	Wireshark

3.2 Agile Methodology

Agile methodology is a software development approach that focuses on delivering high-quality software quickly and efficiently through short development cycles, regular collaboration, and a focus on meeting customer needs [11]. It is designed to be adaptable and responsive to change, allowing teams to pivot and respond to changes in requirements or challenges.

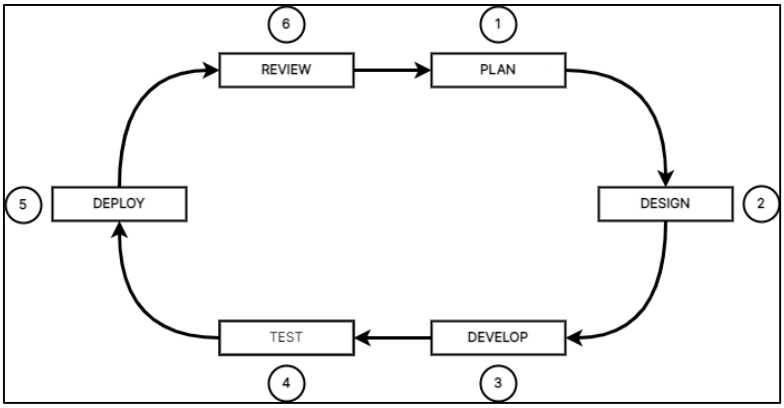

Figure 1: Diagram of Agile Methodology

3.3 Kanban Framework

Kanban is an agile framework that originated in the manufacturing industry and has been adapted for use in software development and other fields [12]. It helps teams improve efficiency and effectiveness by visualizing and managing workflows, identifying bottlenecks and obstacles, and using continuous improvement and collaboration principles. Figure 2 shows an example of a Kanban Framework that is used in the development of the system.

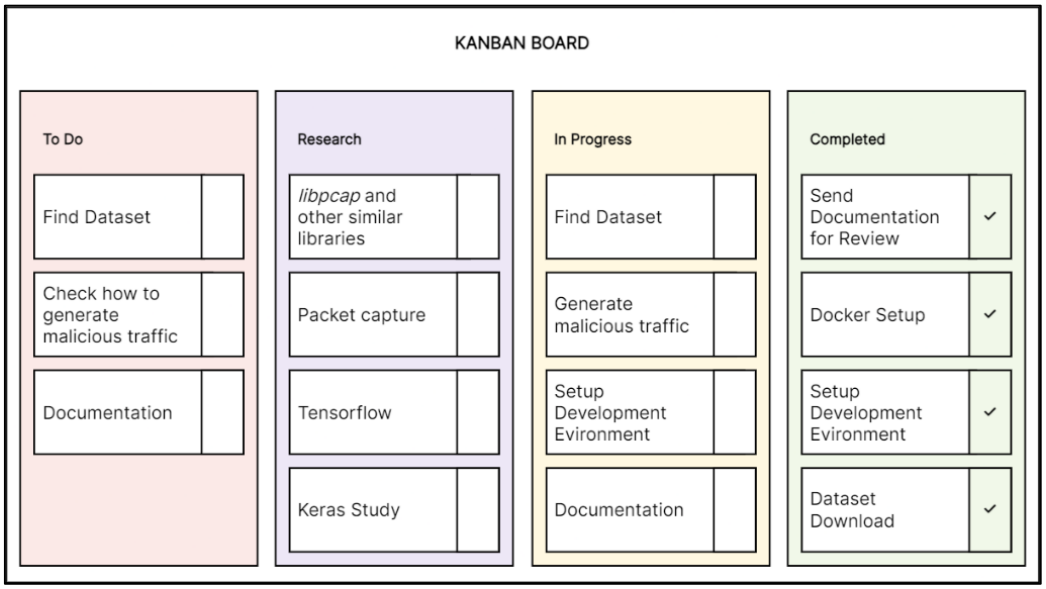

Figure 2: Kanban Framework

3.4 Methodology of the development

The Agile Methodology is the research framework used for this development. It is a software development approach that emphasizes collaboration, flexibility, and iteration. It follows the principles of the Agile Manifesto, valuing individuals and interactions over processes and tools and working software over comprehensive documentation.

3.4.1 Planning the objectives and requirements of the system

The first step in creating an effective IDS for a homelab environment requires identifying objectives and requirements, determining goals, scope, and constraints, identifying types of attack to be detected, and gathering information through user survey or research. IDS should detect DDoS attacks and provide protection for homelab environments.

3.4.2 Design the system architecture of the system

The second step in creating an ANN-based IDS for a homelab is to design the system architecture, which includes determining the ANN architecture & algorithm, selecting software and technologies, identifying the training dataset, and defining the input and output data The CICIDS2017 dataset is selected for its updated DDoS dataset definitions. TensorFlow is used as the AI framework, and the system will be deployed in containers. Data will be received and analysed using a packet sniffer library and log parser, and the web interface will be designed using Streamlit. The system's effectiveness can be tested through simulations or experiments.

3.4.3 Development of the system

The third step in creating an ANN-based intrusion detection system for a homelab environment is the development, that involves implementing the system design, coding, training the ANN and integrating the system components. TensorFlow and Docker are being used for coding, training and system deployment while Nginx, Flask, SQLite and Bootstrap are being used to create a web interface, to let users easily interact with the system and its features.

3.4.4 Testing of the system

The fourth step in creating an ANN-based IDS for a homelab includes testing to ensure the system is detecting attacks effectively. Testing includes unit, integration, and acceptance testing. The system is tested on a Raspberry Pi 4 due to its popularity, low cost, and small form factor. System performance is tested by conducting DDoS attacks and recording detection rate, error rate of ANN model, as well as system container performance. Hardware performance is also recorded. ANN model is evaluated by testing it on a different dataset than the training dataset to ensure it generalizes well.

3.4.5 Implementation and deployment of the system

After thorough testing, the system can be deployed in the target environment by loading it with the services and applications commonly run on a homelab device, including any necessary dependencies or libraries. The device should be placed in a location with a medium network traffic density to ensure optimal system performance.

3.4.6 Review and analysis of the system

To improve and update the system, review and analyze NIDS performance after deployment by collecting data on accuracy and efficiency, identifying potential issues, and evaluating system performance and objectives using metrics. Improve ANN performance by retraining the model with revised variables and better data if needed, and address any system flaws by conducting further testing to ensure effective operation.

4. System Analysis and Design

4.1 Requirement Analysis

Requirement analysis is the process of identifying and understanding the needs and desires of users for an application to be developed or updated. It involves analyzing, documenting, validating, and

managing the software or system requirements to ensure that the final system meets the user's needs and functions as intended. This includes both functional and non-functional requirements.

4.1.1 Functional Requirement

Functional requirements outline the core functionality of a system, including its features and focus on user needs. These requirements specify the intended actions and behaviors of the system under various circumstances.

Table 3: Functional Requirements of the System

Functional Requirements	Functionalities
Intrusion Detection System	1. Continuously monitor network traffic for potential intrusions 2. Accurately identify different types of intrusions on the network. 1. 3. Capture and analyze traffic from the network.
Artificial Neural Network Model	1. Process and analyze large volumes of network traffic in real-time. 2. Accurately identify and classify different types of intrusions based on patterns in network traffic. 3. Handle high levels of noise and variability in network traffic without degrading performance. 4. Classify intrusions with minimal false positives.
Graphical User Interface	1. Provide an intuitive and user-friendly interface for security personnel to interact with the intrusion detection system. 2. Display real-time information about the state of the network and any identified intrusions, including visualizations and alerts. 3. Display detailed reports and other relevant information about past intrusions, including the ability to search and filter data. 4. Responsive and support different devices and screen sizes.

4.12 Non-Functional Requirement

Non-functional requirements are specifications that define the overall behavior, functionality, and characteristics of a system. These requirements do not relate to the system's specific features or functionality, but rather focus on how the system operates and performs.

Table 4: Non-Functional Requirements of the System

Non-Functional Requirements	Functionalities
Performance	1. Handle high volumes of traffic and requests without degrading performance 2. Responsive and low latency for all users 3. Handle concurrent users and transactions without issues 4. Minimize the impact on network performance and bandwidth usage.
Usability	1. Easy to use and navigate for all users 2. Clear and intuitive user interface 3. Well-organized and consistent in its layout and design
Availability	1. High uptime and be available for use at all times 2. Robust failover and recovery mechanisms to ensure continuity of service 3. Handle unexpected events or failures without disrupting service
Security	1. Protect against unauthorized access or tampering 2. Measures in place to prevent data breaches or leaks 3. Comply with relevant security standards and regulations
Confidentiality	1. Protect the confidentiality of sensitive or personal data 2. Prevent unauthorized access or disclosure of sensitive information 3. Safeguards to prevent data breaches or leaks
Integrity	1. Maintain the integrity of data and ensure that it is accurate and consistent 2. Controls in place to prevent data corruption or tampering 3. Mechanisms to detect and correct errors or inconsistencies in data.

4.2 Use Case Diagram

The use case diagram is a visual representation of the functionalities and interactions within a system, illustrating the various actors involved and their interactions with the system. Figure 3 shows use case diagram consists of five main use cases, including Classify Network Traffic, Login, Packet Monitoring and Attack Detection. In this application, the actor is the user, who can authenticate into the system, monitor packets, and receive warnings about attacks.

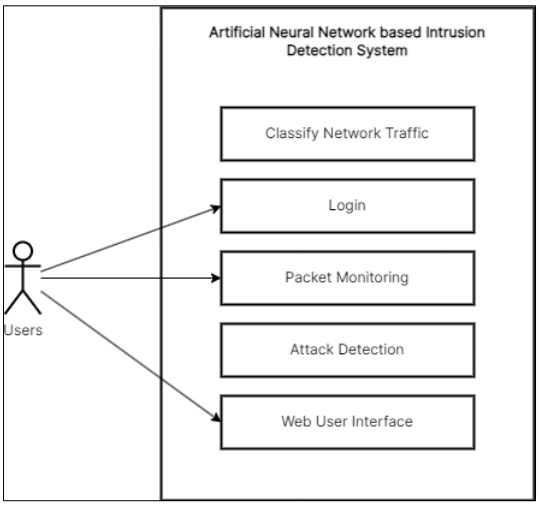

Figure 3: Use Case Diagram of the proposed system

4.2 Sequence Diagram

A sequence diagram is a visual representation of the interactions between objects in a system as illustrated in Figure 4. It is organized in a chronological sequence and illustrates the order in which messages are exchanged between objects. The system starts with the process of system in detecting and alerting the user of any potential intrusions on a network. The user sends a login request to the IDS, which verifies their credentials and grants or denies access. IDS monitors network packets, sending them to an ANN model for analysis and intrusion detection.

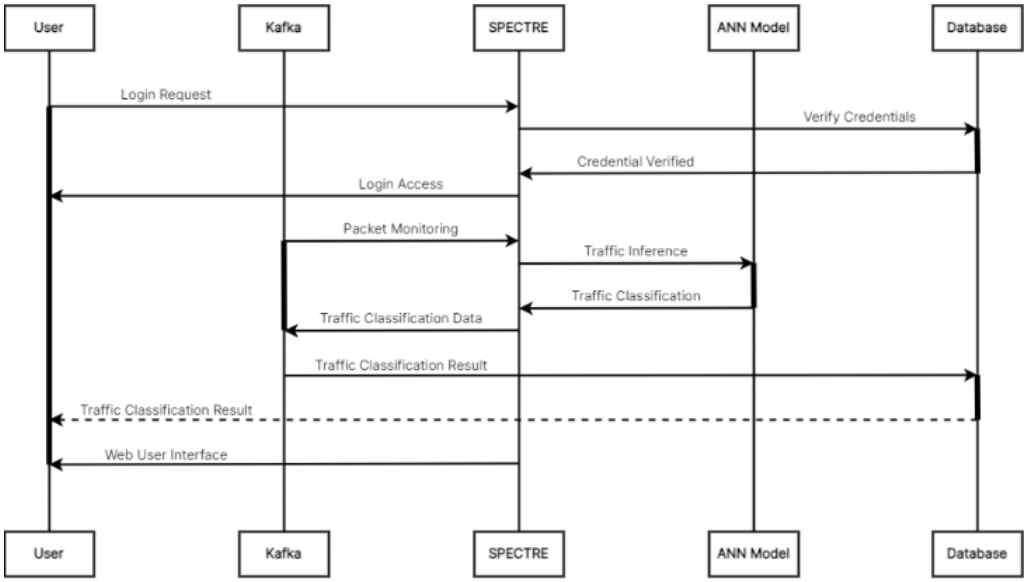

Figure 4: Sequence Diagram of the system

4.3 Activity Diagram

Activity diagrams are a type of flowchart that depict the flow of activities or actions within a system. They are often used to illustrate the steps involved in a single operation or process and show how the events in that process are related to one another.

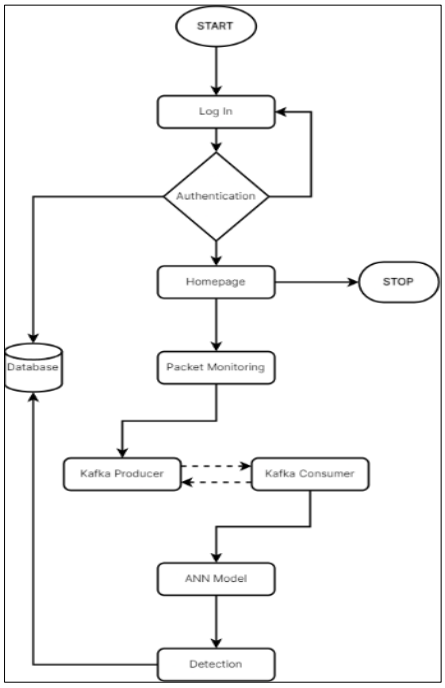

Figure 5: Activity Diagram of the system

Figure 5 illustrates the user activity diagram of the system. Users set up their credentials during the deployment of the system on their homelab and log in to their system dashboard through the given IP address. If the credentials match with the database entry, they are allowed access to the system dashboard. The system continuously monitors network packets in the system and passes them to a Kafka Producer. The ANN model conducts inference on the received data and prints out the predictions and F1 scores. The results are then pushed to the database for visualization purposes. The system continues in a perpetual cycle of monitoring until the user manually turns off the system or removes it from their homelab.

4.4 System Architecture Diagram

A System Architecture Diagram is a visual representation of the components and their relationships within a system. It shows how the system is structured and how the components interact with each other. The components can be hardware, software, or a combination of both

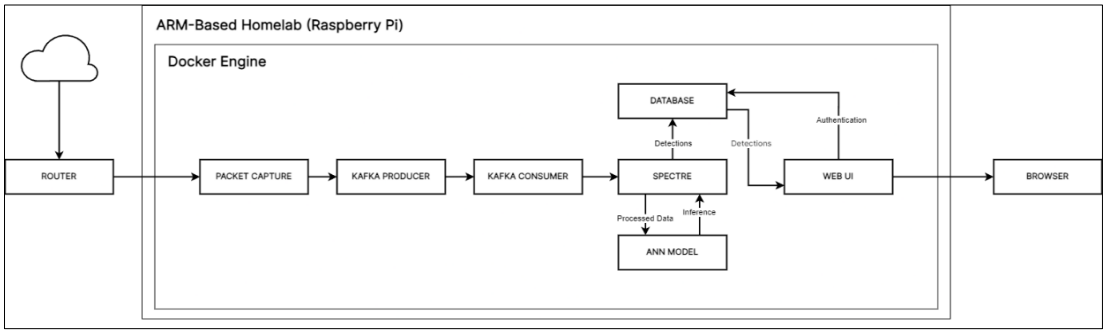

Figure 6: System Architecture Diagram of the proposed system

Figure 6 explains the system architecture and the operation for the proposed system. First, the packet capture module intercepts packets and sends the data to the Network Intrusion Detection System (NIDS). Second, the data is then passed to Kafka, ensuring seamless data flow. In the third step, data preprocessing occurs, including extracting relevant information and performing necessary transformations. The processed data is transmitted through a Kafka consumer. The fourth step involves the ANN model conducting inference and detecting DDoS activities, generating results and an F1 score. These results are sent to a database in the fifth step, providing a central repository for storing attack data. Lastly, the sixth step involves visualizing the data in a web user interface, allowing real-time monitoring and historical analysis, empowering users to respond promptly to detected DDoS activities.

4.5 Interface Design

Effective UI design revolves around creating a seamless and intuitive interface that allows users to accomplish their desired tasks effortlessly and efficiently. The user interface for the system was developed with using the Streamlit framework for its ease of use and compatibility with AI projects. Due to the lightweight nature of the framework, it is able to provide a intuitive user interface without impacting the homelab's performance. Figure 7 shows the wireframes of the graphical user interface of the proposed system.

Figure 7: Web Interface Homepage

4.6 System Testing

System testing is a type of software testing performed on a whole integrated system to verify if it complies with the applicable requirements. This project will use functional testing to assess the features and functioning of the proposed application.

4.6.1 Test Plan

A Test Plan is a well-structured document that specifies the test strategy, objectives, timeframe, estimates, deliverables, and resources required for software testing. The Test Plan assists the developer

in determining the amount of effort required to confirm the quality of the application. The test plan's goal is to ensure that each module functions properly and achieves the desired results. Tables 5.1 to 5.3 illustrate the test plans for all the functionalities included in the proposed system. The testing process was carried out by a group of dedicated homelab users, ensuring a thorough evaluation of the system's performance.

Table 5: Testing Results for Data producer script "*producer.py*"

No.	Functions	Test Case	Expected Output	Actual Output
1	Kafka Producer	Kafka configuration is not found	Kafka consumer instance is not created	Pass
		Handshake message is not received from Kafka consumer	Kafka pipeline for pre-processed data is not created and an error message is shown	Pass
		Producer is not able to push data to Consumer	Producer instance is flushed and closed	Pass
2	Kafka Consumer	Kafka configuration is not found	Kafka consumer instance is not created	Pass
		Handshake message is not received from Kafka producer	Kafka pipeline for pre-processed data is not created and an error message is shown	Pass
3	Data Preprocessing	CSV input is not found	Data preprocessing does not begin, and error is not thrown	Pass
		PCA features are not able to be derived	Error is thrown and data preprocessing is stopped	Pass
		Pre-processed data is not able to be pushed to a dataframe	Error is thrown and the preprocessing starts again	Pass
		Label binarization is not able to be performed	Label binarization is skipped and data is pre-processed	Pass

Table 6: Testing Results for system homepage

No.	Functions	Test Case	Expected Output	Actual Output
1	Kafka Producer	Kafka configuration is not found	Kafka consumer instance is not created	Pass
		Handshake message is not received from Kafka consumer	Kafka pipeline for preprocessed data is not created and an error message is shown	Pass
		Producer is not able to push data to Consumer	Producer instance is flushed and closed	Pass
2	Kafka Consumer	Kafka configuration is not found	Kafka consumer instance is not created	Pass
		Handshake message is not received from Kafka producer	Kafka pipeline for preprocessed data is not created and an error message is shown	Pass
3	ANN Model	ANN Model is not able to load	Data preprocessing does not begin and error is not thrown	Pass
		Inference is not able to be made	The system throws an error message	Pass
4	Database	Table "*predictions*" does not exist	Table "*predictions*" is created with the necessary values.	Pass
		Values of the columns are mismatched	Data is not pushed to the database	Pass

Table 7: Testing Results for system homepage

No.	Functions	Test Case	Expected Output	Actual Output
1	Registration	Incomplete data input	An alert message will display if the text field is empty.	Pass
		Unique username	An alert message will display if the username already exists in the database.	Pass
		Complete registration form	Successfully register user with Information and redirected to home page.	Pass
2	Login	Incomplete data input	An alert message will display and ask the user to complete the login form.	Pass
		Complete input with invalid username or password	Alert message will display, and login request is rejected.	Pass
		Completed login form	Login successfully and redirected to the homepage.	Pass
3.	Dashboard	Date selected has not logs in "*predictions*" table	Table shows empty data and line graph shows an error message	Pass
		Date selected has logs in "*predictions*" table	Table shows data and a line graph is drawn	Pass
		"*Clear Database*" button is selected	Data in table "*predictions*" is cleared	Pass
		Kafka service is not able to be reached	Kafka metrics shows the error message	Pass
		No data is found in table "*predictions*"	Attack metrics, table and line graph does not display data	Pass

5. Implementation

5.1 Docker Container Configuration

Docker containers are used to deploy important components of the system. Containers are used to deploy Kafka, Spectre scripts and Streamlit framework. When deploying multiple containers for an application stack, "*docker-compose.yml*" is used to declare services and "*Dockerfile*" is used to build a Docker image**.**

Docker Compose file, "*docker-compose.yml*" is used to set up a Kafka environment along with Zookeeper. The first service defined is "*zookeeper*", which pulls the "*bitnami/zookeeper*" image and exposes port "*2181*" for communication. The second service is "*kafka*", which pulls the "*bitnami/kafka*" image and exposes port "*9092*" for communication. The next two services, "*spectre_producer_anomaly_detector*" and "*spectre_streamlit_server*", build Docker images using the specified "*Dockerfile*" and "*Dockerfile.streamlit*" respectively. The "*spectre_streamlit_server*" service also exposes port "*8501*" for accessing the Streamlit server and "*volumes*" section defines a named volume called "*kafka-data*" that will be used by the "*kafka*" service for persistent storage. Figure 8 shows the Docker Compose file that is used to define and run the services that are essential for the system.

```yaml
#-------------------------------------------------------------------------
#    Kafka Environment: kafka-docker with Kafka and Zookeeper
#-------------------------------------------------------------------------

version: '2'
services:
  zookeeper:
    image: 'bitnami/zookeeper:latest'
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
  kafka:
    image: 'bitnami/kafka:latest'
    ports:
      - '9092:9092'
    environment:
      - KAFKA_BROKER_ID=1
      - KAFKA_LISTENERS=PLAINTEXT://:9092
      - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
      - KAFKA_CONTROLLER_LISTENERS=PLAINTEXT://:9093
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_ENABLE_KRAFT=no
      - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092,PLAINTEXT://:9093
      - KAFKA_CFG_REPLICA_FETCH_MAX_BYTES=10485760
      - KAFKA_CFG_MAX_REQUEST_SIZE=10485760
    volumes:
      - kafka-data:/bitnami/kafka
  spectre_producer_anomaly_detector:
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - kafka
    volumes:
      - prototype:/prototype
  spectre_streamlit_server:
    build:
      context: .
      dockerfile: Dockerfile.streamlit
    ports:
      - '8501:8501'
    depends_on:
      - spectre_producer_anomaly_detector
    volumes:
      - prototype:/prototype
      #- ~/prototype:/prototype # Use this for Raspberry Pi Deployement
volumes:
  kafka-data:
```

Figure 8: Docker Compose file

The Dockerfile sets the working directory as "*/prototype*", copies the "*requirements.txt*" file into the container, and installs the dependencies listed in "*requirements.txt*" using the "*pip3*" command with the "*--no-cache-dir"* option. The "*EXPOSE*" instruction exposes port "*9092*". The "*CMD*" instruction specifies the command to run when the container starts, which launches a shell command that runs "*producer.py*" and "*anomaly_detector.py*" using Python. Figure 9 shows the Dockerfile that is used to run the system Python code.

```dockerfile
#-------------------------------------------------------------------------
#    Dockerfile: spectre_producer_anomaly_detector
#-------------------------------------------------------------------------

FROM python:3.9-slim-bullseye
# Use arm64v8 for Raspberry Pi Deployment
#FROM arm64v8/python:3.9-slim-bullseye

# Set the working directory
WORKDIR /prototype

# Copy the requirements.txt file into the container
COPY requirements.txt .

RUN pip3 install --upgrade pip

# Install any necessary dependencies
RUN pip3 install --no-cache-dir -r requirements.txt

# Expose the Kafka server port and Streamlit port
EXPOSE 9092

# Run producer.py and anomaly_detector.py
CMD ["sh", "-c", "python producer.py & python anomaly_detector.py"]
```

Figure 9: Dockerfile file

The "*Dockerfile.streamlit*" builds a Docker image for running a Streamlit server. It starts with a base image "*python:3.9-slim-bullseye*" and sets the working directory as "*/prototype*". The "*apt-get*" command installs required dependencies "*sqlite3*" and "*libsqlite3-dev*". It copies the "*requirements.txt*" file into the container and installs the dependencies listed in "*requirements.txt*" using the "*pip3*" command with the "*--no-cache-dir*" option. The "*EXPOSE*" instruction exposes port "*8501*" for the Streamlit server. The "*CMD*" starts the Streamlit server by running the "*streamlit run*" command with the "*homepage.py*" file and additional arguments to configure the server port and run it in headless mode. Figure 10 shows the Dockerfile that is used to define and run the Streamlit web server for the system.

```
#-------------------------------------------------------------------------------
#   Dockerfile.streamlit: spectre_streamlit_server
#-------------------------------------------------------------------------------

FROM python:3.9-slim-bullseye

# Use arm64v8 for Raspberry Pi Deployment
#FROM arm64v8/python:3.9-slim-bullseye

# Set the working directory
WORKDIR /prototype

# Install sqlite3
RUN apt-get update && apt-get install -y sqlite3 libsqlite3-dev

# Copy the requirements.txt file into the container
COPY requirements.txt .

RUN pip3 install --upgrade pip

# Install any necessary dependencies
RUN pip3 install --no-cache-dir -r requirements.txt

# Copy the homepage.py file into the container
#COPY homepage.py .

# Expose the Streamlit port
EXPOSE 8501

# Start the Streamlit server
CMD ["streamlit", "run", "homepage.py", "--server.port", "8501", "--server.headless", "True"]

```

Figure 10: Streamlit Dockerfile file

5.2 System Code

There are two Python scripts that work together to make the system work. The Python script "*producer.py*" is responsible for preprocessing and sending data to ANN model while "*anomaly_detector.py*" is responsible to process the inference and predictions done by the ANN model and push it to a database.

5.2.1 Kafka Producer

The "*producer.py*" Python code reads a CSV file, preprocesses the data, and sends it to a Kafka topic for anomaly detection. The Confluent Kafka library is used to create a Kafka producer and a Kafka consumer. The producer configuration is defined using a dictionary called "*producer_conf,*' which specifies the bootstrap servers using the "*bootstrap.servers*" key. The consumer configuration is defined using a dictionary called "*handshake_consumer_conf*", which specifies the bootstrap servers and additional parameters such as the consumer group ID, session timeout, and auto offset reset. The script performs a handshake with the consumer, and the producer sends a ready message to the consumer using the "*produce()*" method once the connection is established. The script iterates through the chunks of a CSV file, preprocesses each chunk using the "*prod_datapreprocess*" function, and sends the preprocessed data to the Kafka topic "*detect_anomalies*" line by line. The preprocessed data is serialized into a comma-separated string format, and the Kafka producer is used to produce the serialized data to the topic "*detect_anomalies*" using the "*producer.produce()*" method.

```
#-------------------------------------------------------------------------------
#   Data preprocessing function
#-------------------------------------------------------------------------------

# Define a function to preprocess the data
def prod_datapreprocess(df, dimensions_num_for_PCA=7):

    dimensions_num_for_PCA = 7

    # Function to clean the dataset by removing NaN, inf, and -inf values
    def clean_dataset(df):
        # ... (Cleaning dataset)
        return df[indices_to_keep]

    # Function to get PCA feature names
    def get_PCA_feature_names(num_of_pca_components):
        # ... (Process PCA feature names)
        return feature_names

    # Preprocess the dataset
    # Performing PCA
    pca = PCA(n_components=dimensions_num_for_PCA)
    principal_components = pca.fit_transform(df_scaled)

    # Combine the principal components with the original labels
    df_final = pd.concat([df_pc, df_labels], axis=1)

    # Perform label binarization. Converts "ANOMALY" = 1 and "BENIGN" = 0.
    lb = LabelBinarizer()
    df_final['label'] = lb.fit_transform(df_final['label'])

    # Split the dataset into features (X) and labels (y)
    X = df_final.drop(['label'], axis = 1)

    # Returns features(X)
    return X

```

Figure 11: Data preprocessing function

```
#-------------------------------------------------------------------------------
#   Send preprocessed data to Kafka consumer
#-------------------------------------------------------------------------------

# Iterate through the chunks of the CSV file
for chunk in X:
    # Preprocess the current chunk
    X = prod_datapreprocess(chunk)

    # Iterate through the preprocessed data and send it to the Kafka producer line by line
    for i, row in X.iterrows():
        serialized_data = ','.join(map(str, row.values))
        print(f"Serialized data: {serialized_data}")
        producer.produce('detect_anomalies', serialized_data)
        time.sleep(1.5)

# Flush the producer to ensure all messages are sent
producer.flush()

```

Figure 12: Sending data to Kafka Consumer

5.2.2 DDoS Detector and Consumer

The "*anomaly_detector.py*" Python code consumes Kafka messages, processes them using a pre-trained TensorFlow model for anomaly detection, and stores the results in an SQLite database. The script initializes a Console object from the "*rich*" library for more visually appealing output, connects to an SQLite database, creates a table called "predictions" if it does not exist, and commits the changes. The producer configuration for Kafka is defined, and a Consumer object is created that subscribes to the "handshake" topic. The script performs a handshake with the consumer, and the producer sends a ready message to the consumer using the "*produce()*" method once the connection is established. The "*on_message*" function processes received messages from a Kafka consumer, and the predicted labels are set to 1 if any prediction value exceeds the threshold of 0.7, indicating an anomaly, and 0 otherwise. The F1 score is calculated based on the true and predicted labels. The code generates a table named "*Detection*" using the Rich library to display the prediction, result, and F1 score values. The prediction, result, F1 score, and current timestamp are inserted into an SQLite database table named "*predictions*" using an "*SQL INSERT*" statement.

```
#-------------------------------------------------------------------------------
#   Database setup
#-------------------------------------------------------------------------------

db_path = "database/predictions.db"
conn = sqlite3.connect(db_path)
c = conn.cursor()
c.execute('''
   CREATE TABLE IF NOT EXISTS predictions (
       id INTEGER PRIMARY KEY AUTOINCREMENT,
       prediction TEXT,
       result TEXT,
       f1_score TEXT,
       timestamp DATETIME
   )
''')
conn.commit()


```

Figure 13: Database creation

```
#-------------------------------------------------------------------------------
#   Kafka consumer and producer setup (cont.. )
#-------------------------------------------------------------------------------

# Create a Kafka consumer instance for handshake
handshake_consumer = Consumer(handshake_consumer_conf)
handshake_consumer.subscribe(['handshake'])

# Initialize timeout counter and limit for handshake
timeout_counter = 0
timeout_limit = 50

# Perform handshake with the producer
while True:
    msg = handshake_consumer.poll(1.0)
    if msg is None:
        timeout_counter += 1
        if timeout_counter >= timeout_limit:
            print("═══════════════════════════════")
            print("CONNECTION FAILURE")
            print("═══════════════════════════════")
            exit(1)
        continue
    if msg.error():
        print(f"Handshake consumer error: {msg.error()}")
    else:
        handshake_msg = msg.value().decode('utf-8')
        if handshake_msg == 'READY':
            print("═══════════════════════════════")
            print("CONNECTION ESTABLISHED")
            print("═══════════════════════════════")
            break

# Subscribe to the 'detect_anomalies' topic
consumer.subscribe(['detect_anomalies'])


```

Figure 14: Kafka Handshake

```
#──────────────────────────────────────────────────────────────────────────────
#   Message processing
#──────────────────────────────────────────────────────────────────────────────

def on_message(msg):
    ...
    if len(received_data_buffer) == 7:
        ...
        prediction = model.predict(X_received, verbose=0)
        threshold = 0.7
        ...
        f1 = f1_score(y_true, y_pred)
        ...
        table = Table(title="Detection")
        ...
        console.print(table)
        ...
        c.execute('''
            INSERT INTO predictions (prediction, result, f1_score, timestamp) VALUES (?, ?, ?, ?)
        ''', (str(prediction), result, str(f1), current_time))
        conn.commit()
    else:
        ...

```

Figure 15: Message processing function

5.2.3 Homepage

The python code explains the web user interface used for the system, developed using the Streamlit framework. The dashboard has three tabs: "Dashboard," "Account Management," and "Developer Options." The "Dashboard" tab displays DDoS attack predictions using the "*display_predictions*" function, and the data can be refreshed or deleted. The "Account Management" tab includes an account expander to register new users using the "*register_user*" function. The "Developer Options" tab includes a developer area expander. The login page is implemented using the "*login_page*" function, which verifies the entered username and password and sets session state variables accordingly. The "*register_user*" function renders a registration form on the dashboard, prompting the user to enter a username, password, and confirm the password.

```
#──────────────────────────────────────────────────────────────────────────────
#   auth.py
#──────────────────────────────────────────────────────────────────────────────

import streamlit as st
import pandas as pd
import sqlite3
from confluent_kafka.admin import AdminClient
import plotly.express as px
import altair as alt
from auth import create_users_table, add_user, verify_password, user_exists

# Connect to the SQLite database
db_path = "database/predictions.db"
conn = sqlite3.connect(db_path)
c = conn.cursor()

# Login form
def login_page():
    # Login page Code .....

# User Register Page
def register_user():
    # User Registration code .....


# Function to fetch data from the database and display it
def display_predictions():
    # Dashboard Code .....


def dashboard_page():
    # Add a collapsible sidebar for the refresh button
    with st.sidebar:
        #Sidebar code ....

    # dashboard title
    st.title("🧠 SPECTRE DASHBOARD")
    st.header(f"Hello, {st.session_state.username}")
    st.caption("A lightweight solution for DDoS Detection")

    # Create a placeholder for the line chart
    line_chart_placeholder = st.empty()

    tab1, tab2, tab3 = st.tabs(["Dashboard", "Account Management", "Developer Options"])

    with tab1:
        display_predictions()


    with tab2:
        account_expander = st.expander(label='Register Account')

    with tab3:
        dev_expander = st.expander(label='Developer Area')


if "logged_in" not in st.session_state:
    st.session_state.logged_in = False

if not st.session_state.logged_in:
    login_page()
else:
    dashboard_page()
```

Figure 16: Homepage Code

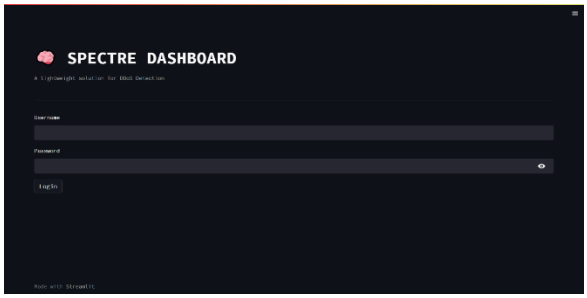

Figure 17: Dashboard login page

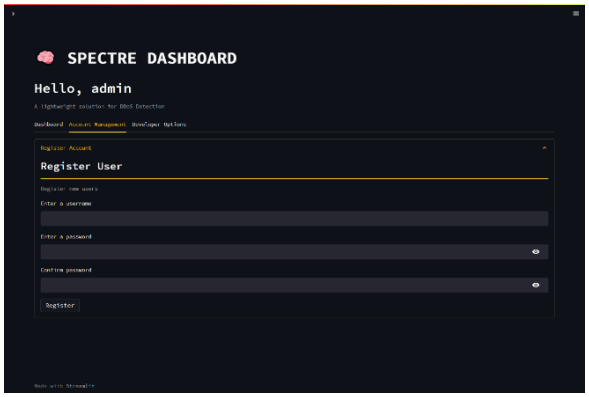

Figure 18: User registration form

Figure 19: Dashboard homepage

6. Result and Discussion

6.1 User Acceptance Testing

User Acceptance Testing (UAT) is a method of testing in which the end-user or customer verifies and accepts the software system before it is moved to the production environment. Table 8 shows the results and feedback based on the user system evaluation that five homelab users have conducted. Due to time limitations, a security professional and four amateur homelab users are involved in this testing. For the user interface aspect, most users have felt that it was easy to use and intuitive while some where unsatisfied with the layout and the inability to change the data viewing format. Next, Table 9 shows the evaluation of the system functions. Testers were satisfied with most of the system functions including the Kafka consumer and producer, deployment, and DDoS detection. Finally, Table 10 provides the results for the system experience of the tester while using the system. Most of the users were content with the deployment and performance of the system while multiple testers are considering deploying the system in their homelab.

Table 8: Result of system user interface evaluation

No.	Features	Ranking					Total
		1 (Poor)	2	3	4	5 (Very Good)	
1.	Easy to use and Intuitive					5	5
2.	Content Navigation				4	1	5
3.	Layout of Content			1	4		5
4.	Content Viewing			1	2	2	5
5.	Interface Design			1	3	1	5
6.	Attack Visualization			2	3		5

Table 9: Result of system function evaluation

No.	Features	Ranking					Total
		1 (Poor)	2	3	4	5 (Very Good)	
1.	User Registration			2	2	1	5
2.	Login Function					5	5
3.	DDoS Detection				3	2	5
4.	Kafka Consumer and Producer				1	4	5
5.	Deployment					5	5

Table 10: Result of system experience function evaluation

No.	Features	Ranking					Total
		1 (Poor)	2	3	4	5 (Very Good)	
1.	Ease of Deployment				1	4	5
2.	Advantage of Containerization				1	4	5
3.	Performance Impact				4	1	5
4.	Detecting Effectiveness				2	3	5
5.	Achievement of goals			1	1	3	5
6.	Implementation Consideration				4	1	5

6.2 DDoS Detection

The DDoS detection capability of the Artificial Neural Network (ANN) model can be evaluated using metrics such as Confusion Matrix, Precision, F1-Score, Recall and Accuracy. Table 11 displays the metrics' values.

Table 11: ANN Model Metrics

No	Metrics	Value
1.	Confusion Matrix	18409 1010 789 24935
2.	Precision	0.96
3.	F1-Score	0.97
4.	Recall	0.97
5.	Test Set Accuracy	0.96

The Confusion Matrix reveals the model's performance in classifying DDoS attacks and normal traffic. In this case, the matrix shows that out of a total of 19,209 instances of normal traffic, the model correctly identified 18,409, but misclassified 1,010 as DDoS attacks. Similarly, out of 25,924 instances of DDoS attacks, the model accurately detected 24,935, while incorrectly labeling 789 as normal traffic.

Precision, a measure of the model's ability to correctly identify positive instances, is calculated to be 0.96. This indicates that 96% of the instances classified as DDoS attacks by the model were indeed true positives. Recall, also known as sensitivity or true positive rate, measures the model's ability to identify all positive instances. With a recall score of 0.97, the model successfully identified 97% of the actual DDoS attacks present in the dataset. The F1-score, which combines precision and recall into a single metric, is calculated to be 0.97. This score provides an overall assessment of the model's

performance by considering both false positives and false negatives. Lastly, the test set accuracy of the ANN model is reported to be 0.96, indicating that the model achieved an overall accuracy of 96% in correctly classifying instances from the test dataset. These metrics collectively demonstrate the effectiveness of the ANN model in detecting DDoS attacks, with high precision, recall, and F1-score, as well as a commendable overall accuracy.

7. Conclusion

Homelabs, personal servers or clusters of servers used at home, have become popular among IT enthusiasts to create their own services without relying on big tech companies. However, botnets and vulnerable databases pose a threat to homelab users without a comprehensive intrusion detection system. The system has succeeded in creating a user-friendly IDS that enables a seamless deployment using containers and eliminating the complexities associated with installing dependencies and configuring the system environment. In addition, the ANN model deployed ensures that genuine DDoS attacks are promptly identified with low false positives and is highly efficient, even on low-powered devices. In conclusion, the system helps the homelab community by providing a comprehensive intrusion detection system that is easy to deploy and efficient in detecting DDoS attacks.

Acknowledgment

The authors would like to thank the Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia for its support.

References

[1] "What is a Homelab and Why Should You Have One?" https://linuxhandbook.com/homelab/ (accessed Nov. 14, 2022).

[2] "What is an Intrusion Detection System (IDS)? - sunnyvalley.io." https://www.sunnyvalley.io/docs/network-security-tutorials/what-is-intrusion-detection-system#how-does-an-intrusion-detection-system-work (accessed Nov. 14, 2022).

[3] "What is a distributed denial-of-service (DDoS) attack? | Cloudflare." https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (accessed Dec. 07, 2022).

[4] "What is a Neural Network? Defined, Explained, and Explored." https://www.forcepoint.com/cyber-edu/neural-network (accessed Nov. 14, 2022).

[5] "5 Amazing Applications of Deep Learning in Cybersecurity." https://www.datto.com/blog/5-amazing-applications-of-deep-learning-in-cybersecurity (accessed Nov. 22, 2022).

[6] "Apache Kafka Tutorial - A Comprehensive Guide - 2023." https://mindmajix.com/apache-kafka-tutorial (accessed Jun. 18, 2023).

[7] "Streamlit 101: An in-depth introduction | by Shail JD | Towards Data Science." https://towardsdatascience.com/streamlit-101-an-in-depth-introduction-fc8aad9492f2 (accessed Jun. 18, 2023).

[8] "Snort IDS/IPS Explained. What - Why you need - How it works - sunnyvalley.io." https://www.sunnyvalley.io/docs/network-security-tutorials/what-is-snort (accessed Nov. 14, 2022).

[9] "What is Suricata - Suricata - Open Information Security Foundation." https://redmine.openinfosecfoundation.org/projects/suricata/wiki/What_is_Suricata (accessed Nov. 14, 2022).

[10] "Suricata: What is it and how can we use it | Infosec Resources." https://resources.infosecinstitute.com/topic/suricata-what-is-it-and-how-can-we-use-it/ (accessed Nov. 14, 2022).

[11] "What Is Agile Methodology? (A Beginner's Guide) • Asana." https://asana.com/resources/agile-methodology (accessed Dec. 31, 2022).

[12] "What Is Kanban? A Beginner's Guide for Agile Teams • Asana." https://asana.com/resources/what-is-kanban (accessed Dec. 31, 2022).