# **ABSTRACT**

Everything is evolving toward IoT (Internet of Things) and online-based in our technological environment. The number of IoT devices and ubiquitous computing systems is growing exponentially. This paper presents the concatenation of two well-known kitsune datasets (ARP MITM and SSDP Flood). Random Forest, decision trees, K-means, Deep Neural Network (DNN), and Bi-LSTM were implemented in different training and testing ratios and layers. Performance measures show that random forests outperform the concatenated dataset. Both the attacks are determined by the given model hence increasing performance and the system will notify in case of any malicious activity.

# **ACKNOWLEDGEMENT**

***In the name of Allah, the most Gracious and the Most Merciful.***

***Peace and blessing of Allah be upon Prophet Muhammad* ﷺ**

First, praise of Allah, for giving us this opportunity, the strength and the patience to complete our FYP finally, after the challenges and difficulties. We would like to thank our supervisor, ***Sir Sharukh Shakil***,for his guidance, motivation, and most his significant contribution in this project, expert \_\_\_***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*** and ***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*** for giving us the opportunity to work on this project. We would also like to thanks our parents for financial and moral support and our friends who have helped and motivated us throughout. May Allah reward them all abundantly. Ameen

# **DEDICATION**

This report is dedicated to PAF-KIET University, our Teacher, our Supervisor, our Parents, our fellow colleagues and the hard-working students of PAF-KIET, with a hope that they will succeed in every aspect of their Academic Career and this project may help them in any aspect of their life.

# **TABLE OF CONTENTS**

[**ABSTRACT** 1](#_Toc136364807)

[**ACKNOWLEDGEMENT** 2](#_Toc136364808)

[**DEDICATION** 3](#_Toc136364809)

[**TABLE OF CONTENTS** 4](#_Toc136364810)

[**LIST OF FIGUERS** 6](#_Toc136364811)

[**LIST OF TABLES** 7](#_Toc136364812)

[**CHAPTER 1** 8](#_Toc136364813)

[1. Introduction 8](#_Toc136364814)

[1.1. Motivations 9](#_Toc136364815)

[1.2. Problem Statement 9](#_Toc136364816)

[1.3. Objectives and Contributions 10](#_Toc136364817)

[1.4. Project Scope 10](#_Toc136364818)

[1.5. Organization of the Report 11](#_Toc136364819)

[**CHAPTER 2** 13](#_Toc136364820)

[2. Literature Review/Process Review 13](#_Toc136364821)

[2.1. Introduction 13](#_Toc136364822)

[2.2. Literature Review 13](#_Toc136364823)

[**CHAPTER 3** 17](#_Toc136364824)

[3. Projects diagrams 17](#_Toc136364825)

[**CHAPTER 4** 22](#_Toc136364826)

[4. Project Tools 22](#_Toc136364827)

[4.1. Used Technologies 22](#_Toc136364828)

[**CHAPTER 5** 26](#_Toc136364829)

[5. Project Planning 26](#_Toc136364830)

[5.1. Project Timeline Summary 26](#_Toc136364831)

[5.2. Project Timeline Details 26](#_Toc136364832)

[**CHAPTER 6** 28](#_Toc136364833)

[6. Project Implementation 28](#_Toc136364834)

[6.1. Dataset 28](#_Toc136364835)

[6.2. Methodology 28](#_Toc136364836)

[6.3. Implementation 33](#_Toc136364837)

[6.4. Results 35](#_Toc136364838)

[**CHAPTER 7** 38](#_Toc136364839)

[7. Model Testing 38](#_Toc136364840)

[7.1. ARP-MITM Attack and It’s Performance 38](#_Toc136364841)

[7.2. SYN-Flood DDoS Attack and It’s Performance 39](#_Toc136364842)

[**CHAPTER 8** 41](#_Toc136364843)

[8. Project Prototype 41](#_Toc136364844)

[8.1. Framework 41](#_Toc136364845)

[8.2. Front-end Working 41](#_Toc136364846)

[**CHAPTER 9** 42](#_Toc136364847)

[9. Conclusion and Future Work 42](#_Toc136364848)

[9.1. Limitation 42](#_Toc136364849)

[9.2. Conclusion 42](#_Toc136364850)

[9.3. Future Works 43](#_Toc136364851)

[References 43](#_Toc136364852)

[Code 44](#_Toc136364853)

# **LIST OF FIGUERS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Figure No.** | **Figure Name** | **Page No.** |
| **1** | **Figure 1** | *Proposed learning architecture* | 17 |
| **2** | **Figure 2** | *Proposed IDS architecture* | 18 |
| **3** | **Figure 3** | *MITM Attack Architecture* | 18 |
| **4** | **Figure 4** | *MITM Attack Architecture* | 19 |
| **5** | **Figure 5** | *SYN Flood DDoS Attack Architecture* | 20 |
| **6** | **Figure 6** | *Front-end of our prototype* | 20 |

# **LIST OF TABLES**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Table No.** | **Table Name** | **Page No.** |
| **1** | **Table 1** | *Summary Of Literature Review* | 16 |
| **2** | **Table 2** | *RF and DT on Concatenated dataset ON 80:20, 70:30 AND 90:10* | 35 |
| **3** | **Table 3** | *AUC-scores of RF and DT for ARP\_MITM dataset* | 35 |
| **4** | **Table 4** | *AUC-scores of RF and DT for SSDP\_FLOOD dataset* | 36 |
| **5** | **Table 5** | *Bi-LSTM on concatenated dataset* | 36 |
| **6** | **Table 6** | *Comparison table between all models* | 36 |
| **7** | **Table 7** | *Model Performance Table on ARP\_MITM* | 39 |
| **8** | **Table 8** | *Model Performance Table on SYN-Flood DDoS* | 40­ |

# **CHAPTER 1**

## Introduction

Everything is evolving toward IoT (Internet of Things) and online-based in our technological environment. Anything can be ordered from anywhere with just one click of the mouse and payment. Everything, including your order receipt and bank account information, is saved online. For instance, modern coffee mugs can connect to the internet, allowing us to monitor things like temperature, caffeine intake, sugar level, etc. From these insights, we can now limit how much sugar and caffeine we consume for our own health. Other than this, the entire world has gone online. The majority of institutions, including hospitals, legal firms, universities, and law enforcement agencies store data someplace on servers or in the cloud, including paying bills, conducting bank transactions, preserving information, and practically everything else has gone online.

Now, while this may alleviate many issues, it also raises security and safety concerns since as everything becomes smarter, it becomes easier to hack and breach. Everyone values their privacy, so this is a really serious problem. Different approaches to intrusion detection have been proposed by numerous people, including the use of artificial neural networks (ANN), support vector machines (SVM), machine learning (ML) algorithms, and many others. ML techniques are still utilized today, but Deep Learning (DL) methods are much more sophisticated and are mostly employed for intrusion detection since they work more quickly and efficiently on massive amounts of data than ML algorithms do.

Making an anomaly-based intrusion detection system with machine learning and deep learning is our suggested approach. Anomaly-based detection generally finds data points, events, actions, and/or observations that differ from the normal behavior of the dataset. Unusual data can point to serious occurrences, like technical errors, or to prospective business opportunities, such as a shift in consumer behavior, harmful activity on the network, or a malicious network packet. An anomaly-based intrusion detection system can therefore warn the user of unknown suspicious behavior. From the KITSUNE dataset package, which includes many datasets for various types of attacks, we used the ARP MITM and SSDP FLOOD datasets. Man-in-the-middle (MITM) attacks are handled by ARP MITM, while denial-of-service (DOS) attacks are handled by SSD FLOOD. To determine which models, perform better on which dataset, we applied Decision Tree (DT), Random Forest (RF), K-means, Deep Neural Network (DNN) and Bi-LSTM to both datasets jointly and individually.

### Motivations

Creating a system that can automatically recognize and react to possible security threats in real time may be the driving force behind an intrusion detection project using machine learning and deep learning approaches. This kind of system may be helpful in a range of contexts, such as:

• Network security: Network traffic may be monitored to spot possible dangers like malware infections or unauthorized access attempts using an intrusion detection system.

• Cybersecurity: System logs and other data sources might be monitored by an intrusion detection system to spot possible security lapses or intrusions.

• Industrial control systems: Critical infrastructures, such as power plants or transportation networks, may be monitored for proper functioning and possible threats using an intrusion detection system.

The overall goal of a machine learning and deep learning-based intrusion detection project is to increase the security and consistency of a system by automatically identifying and responding to possible threats in real time.

### Problem Statement

Everything is evolving toward IoT (Internet of Things) and online-based in our technological environment. Anything can be ordered from anywhere with just one click of the mouse and payment. Everything, including your order receipt and bank account information, is saved online. For instance, modern coffee mugs can connect to the internet, allowing us to monitor things like temperature, caffeine intake, sugar level, etc. From these insights, we can now limit how much sugar and caffeine we consume for our own health. Other than this, the entire world has gone online. The majority of institutions, including hospitals, legal firms, universities, and law enforcement agencies store data someplace on servers or in the cloud, including paying bills, conducting bank transactions, preserving information, and practically everything else has gone online.

Now, while this may alleviate many issues, it also raises security and safety concerns since as everything becomes smarter, it becomes easier to hack and breach. Everyone values their privacy, so this is a really serious problem.

### Objectives and Contributions

Objectives:

• Create a deep learning or machine learning model that is capable of successfully detecting possible threats or intrusions in real-time.

• Deploy the model into a network or system to allow automatic threat detection and mitigation.

• Analyze how well the model performs in identifying various threats or intrusions and make any necessary modifications.

Contributions:

• A system's security and reliability can be increased by a machine learning or deep learning model that automatically recognizes and reacts in real time to possible attacks.

• A novel method of intrusion detection that applies the most recent developments in deep learning and machine learning.

• A better comprehension of how efficient various machine learning and deep learning algorithms are at detecting intrusions.

### Project Scope

* Determine the precise type of attack or incursion (DOS, MITM) that the system is designed to catch, then choose the best machine learning or deep learning techniques to apply.
* Collect and classify relevant data so that the model may be trained and tested. System logs, network traffic data, and other pertinent data may be included in this. In our project, we have used the KITSUNE dataset which contains network packets for MITM and DOS attacks
* Create and train a model that can reliably identify threats or intrusions using machine learning and deep learning approaches. In order to do this, the model must be trained using the prepared data, tuned using the necessary techniques, and then assessed that how well the model performed.
* To make sure the model can effectively detect intrusions or threats, deploy the trained model in a system and test it in a real-world environment.

### Organization of the Report

The organization of the report follows a structured approach to present the research findings and project details in a logical sequence.

Chapter 1 serves as the introduction, providing an overview of the research topic, motivations, problem statement, objectives, project scope, and the organization of the report itself.

Chapter 2 focuses on the literature review or process review, starting with an introduction followed by an in-depth review of relevant existing literature or processes related to the research topic.

Chapter 3 presents project diagrams, which include visual representations such as system architecture diagrams to illustrate the project's structure and components.

Chapter 4 delves into the project tools utilized, discussing the technologies and software frameworks employed during the project development.

Chapter 5 covers project planning, including a summary of the project timeline and detailed timeline information to provide insights into the project's milestones and progress.

Chapter 6 focuses on project implementation, starting with the dataset used, followed by the methodology employed, the actual implementation details, and the results obtained.

Chapter 7 centers around model testing, specifically evaluating the performance of the system under different attack scenarios, such as ARP-MITM attack and SYN-Flood DDoS attack.

Chapter 8 explores the project prototype, discussing the chosen framework and providing an overview of the front-end working of the developed application.

Chapter 9 encompasses the conclusion and future work, highlighting the limitations of the project, summarizing the main findings, and suggesting potential areas for future research and development.

Finally, the report concludes with a list of references used throughout the research and the project. This organization ensures a clear and coherent flow of information, allowing readers to follow the research process, understand the findings, and gain insights into the implemented project prototype.

# 

# **CHAPTER 2**

## Literature Review/Process Review

### Introduction

A literature review is an essential component of academic research that involves a systematic examination and evaluation of existing scholarly works, such as books, articles, and dissertations, related to a specific research topic or question. The process of conducting a literature review entails several steps. First, researchers define their research objective and formulate specific research questions to guide their review. They then conduct a comprehensive search of relevant literature sources, including online databases, libraries, and academic journals. After identifying relevant works, researchers critically analyze and synthesize the information presented in each source, focusing on key themes, theories, methodologies, and findings. This analysis helps identify existing knowledge gaps and inconsistencies, which can inform the direction of the researcher's own study. Finally, the researcher organizes and presents the findings in a coherent and logical manner, often through a written literature review, highlighting the significance and relevance of the existing literature to their research topic. Overall, a literature review serves as a foundation for new research, providing a comprehensive understanding of the current state of knowledge and informing the development of research hypotheses and methodologies.

### Literature Review

The number of IoT devices and ubiquitous computing systems is growing daily. The internet is required for these devices to function, and for them to work effectively together, communication is essential. The amount of data is growing dramatically as a result. Now, there are more attacks on these big data and IoT devices, and it is challenging to identify attacks on big data.

In [1] several methods for the detection of attacks were explored.  The majority of issues can be resolved through supervised learning if the data is substantial and valuable. 80%:20% ratio was used as the train test split ratio. Accuracy, precision, sensitivity, specificity, f1-score, detection rate, and false alarm were used as metrics on Decision Tree, Naïve Bayes, Support Vector Machine (SVM), and Adaboost algorithms. In [2] numerous supervised learning algorithms were applied on the ton-iot dataset. Attacks like scanning, cross-site scripting, ransomware, backdoors, distributed denial of service (DDoS), password cracking, the man in the middle, and injection attacks are all included in the dataset. Preprocessing of the dataset was done by imputing the missing values, converting the categorical features into numerical by one hot encoding technique, class imbalance by using the Synthetic Minority Oversampling Technique (SMOTE), dropping some features like IP address, source port, TimeStamp, and destination port, feature selection by using chi^2 technique, and data normalization by using the min-max technique. 30% of the dataset was used for testing the model and 70% was used for training and validation. As metrics recall, precision, false positive rate (FPR), f1-score and accuracy were applied using various algorithms that were random forest, SVM, AdaBoost, logistic regression, decision tree, Naïve Bayes, k-nearest neighbor, and xgboost. In [3] convolutional neural network (CNN) was used to tackle this problem. UNSW-NB15 dataset was used which contains several types of attacks. Preprocessing was needed for this dataset. Objects were converted into numerical vectors, handled missing values, normalized and reshaped dataset for CNN input, one hot encoding on the label to represent the number of classes, and hyperparameter optimization including regularization techniques, learning rate, optimization algorithm, and batch size. In [4] deep learning techniques were used on the ARP kitsune dataset which contains attacks like the man in the middle(MITM), Dos(DOS), and bot-type attacks. The dataset is already preprocessed. The model was trained on two different ratios (80-20, 70-30). Long Short-Term Memory (LSTM) and CudLSTM algorithms were used having 4 hidden layers with, precision, accuracy recall, and f1-score as metrics. In [5] deep learning technique was used on NSL-KDD and CSE-CIC-IDS2018 datasets. Deep Auto Encoder (DAE) was used for feature selection in which some hidden layers were used and encoder and bottleneck layers and some were used as decoders. The output of DAE was then used as an input for Deep Neural Network (DNN) for tuning and detection. In DAE 20 epochs were used and in DNN 30 epochs were used. ReLU activation function was used in DNN with adam optimization function for tuning the value of the learning rate.

In [6] deep learning technique was used on the UNSW-NB15 dataset. The dataset was cleaned and normalized before sending it for training the model. 15%-15% was used for validation and testing the mode whereas 70% data was used for training. Artificial Neural Networks (ANN), DNN, and LSTM were used to train the model. In ANN one hidden layer was used with 850 neurons, adam as an optimizer, relu as activation function, and softmax as output layer activation. 100 epochs were done with 100 batch size. In DNN three hidden layers were used with 100 neurons, adam as an optimizer, relu as an activation function, and softmax as output layer activation. 100 epochs were done with 100 batch size. In LSTM three layers were used with 128,64,32 neurons respectively. Adam as an optimizer, relu as an activation function, and softmax as the output layer activation were used. 100 epochs were done with 100 batch size. In [7] some supervised algorithms (NB, SVM, DT, RF, KNN) and Feed Forward DNN (FFDNN) were used on UNSW-NB15 and AWID datasets. They used a wrapper-based feature extraction technique to determine the weights or importance of the input through a classification model to evaluate that how features perform. Extra Trees (EU) algorithm was also used in the feature selection process. The dataset was split in 75-25% ratio for train-test split ratio. First, all the models were trained on full dataset without feature extraction. After that, all the algorithms were used with a Wrapper-based Feature Extraction Unit (WFEU).

In [8] deep learning and unsupervised learning was used to detect zero-day attacks on SDN20 dataset. To enhance the performance unsupervised learning with boosting meta-learning was used which can perform like supervised-base detection systems. In [9] deep learning with semantic re-encoding was used to increase the ability of IDS to detect the traffic thus increasing the robustness and accuracy of the IDS method. The probability of this IDS of detecting web character injection network attack is over 99% and for the NSL-KDD dataset 8% detection rate is increased from traditional methods. In [10] two types of IDS (anomaly-based, and signature-based) are merged to create a hybrid IDS called AS-IDS. First, network traffic was collected from the gateway of the IoT device by comparing the packet features. After that preprocessing was done by using Discrete Hessian Eigenmap (DHE), target encoder, and Z-score. For signature-based IDS, Lightweight Neural Network (LNN) was used with Human Mental Search (HMS) in the hidden layer which creates clusters of traffic, and Generalized Suffix Tree (GSF) on the output layer. For anomaly-based IDS, Deep Q-learning was used to identify attacks that are unknown in signature-based IDS. Table 1 summarizes all the related work discussed above.

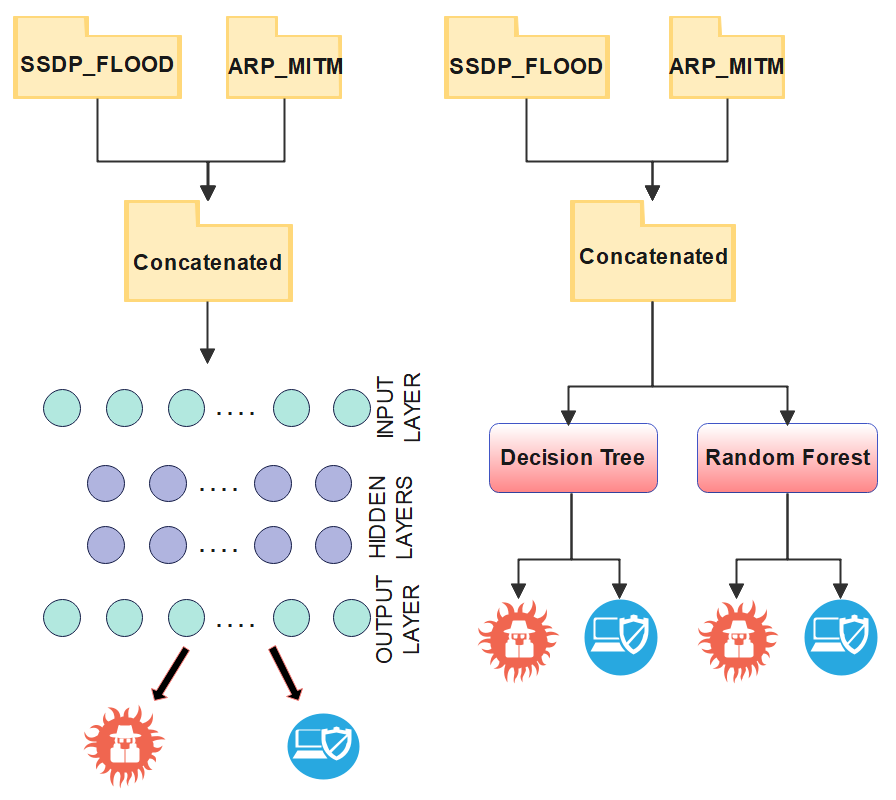
TABLE 1: SUMMARY OF LITERATURE REVIEW

|  |  |  |
| --- | --- | --- |
| **Research Works** | **Approach** | **Contribution** |
| Sai Kiran et al [1] | SVM, NB, DT, AdaBoost | To build a machine learning model which can identify attacks in IoT devices. |
| Gad, Nashat, and Barkat [2] | LR, KNN, NB, SVM, RF, DT, AdaBoost, XgBoost | To build a machine learning model to identify attacks in VANETs for traffic safety. |
| Ashiku and Dagli [3] | CNN | To build a NIDS using deep learning techniques which is flexible and can be used for detecting zero-day attacks. |
| Anwer et al. [4] | LSTM, CudLSTM | To build a IDS using deep learning techniques for smart systems. |
| Kunang [5] | DAE, DNN | To build a NIDS using deep learning techniques for a network. |
| Aleesa et al. [6] | ANN, DNN, LSTM | To build an IDS using deep learning techniques to identify attacks in network traffic. |
| Kasongo and Sun [7] | NB, SVM, DT, RF, KNN, FFDNN | To build a wireless IDS using machine learning techniques for VANETs, IoT devices, Wireless devices, and cyber-physical systems. |
| Zoppi, Ceccarelli, and Bondavalli [8] | Unsupervised anomaly detection algorithms | To build an IDS to deal with zero-day ataacks using unsupervised anomaly detection algorithms. |
| Wu et al. [9] | Algorithm1 +SVM, Algorithm2 +NB, SRDLM including Algorithm1 | To build a IDS using SRDLM to decrease the false-rate alarm and increase performance on big data. |
| Otoum and Nayak [10] | LightNet, HMS, GST, DQN, SNR | To build a hybrid IDS also called AS-IDS to deteck known and unknown attacks. |

# **CHAPTER 3**

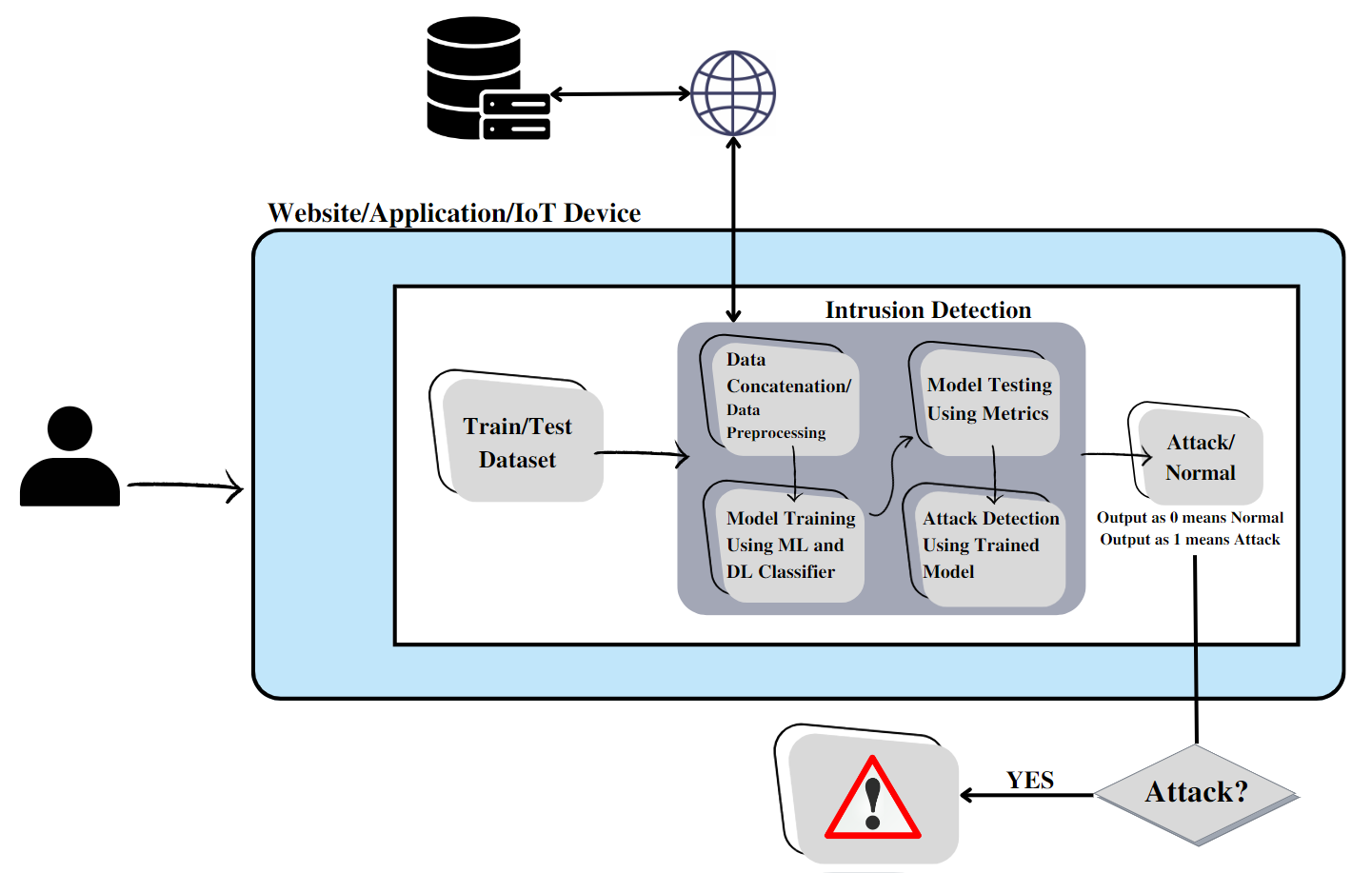
## Projects diagrams

Here are some diagrams, which illustrates that what will be our project or the system is capable to reach the desired results.



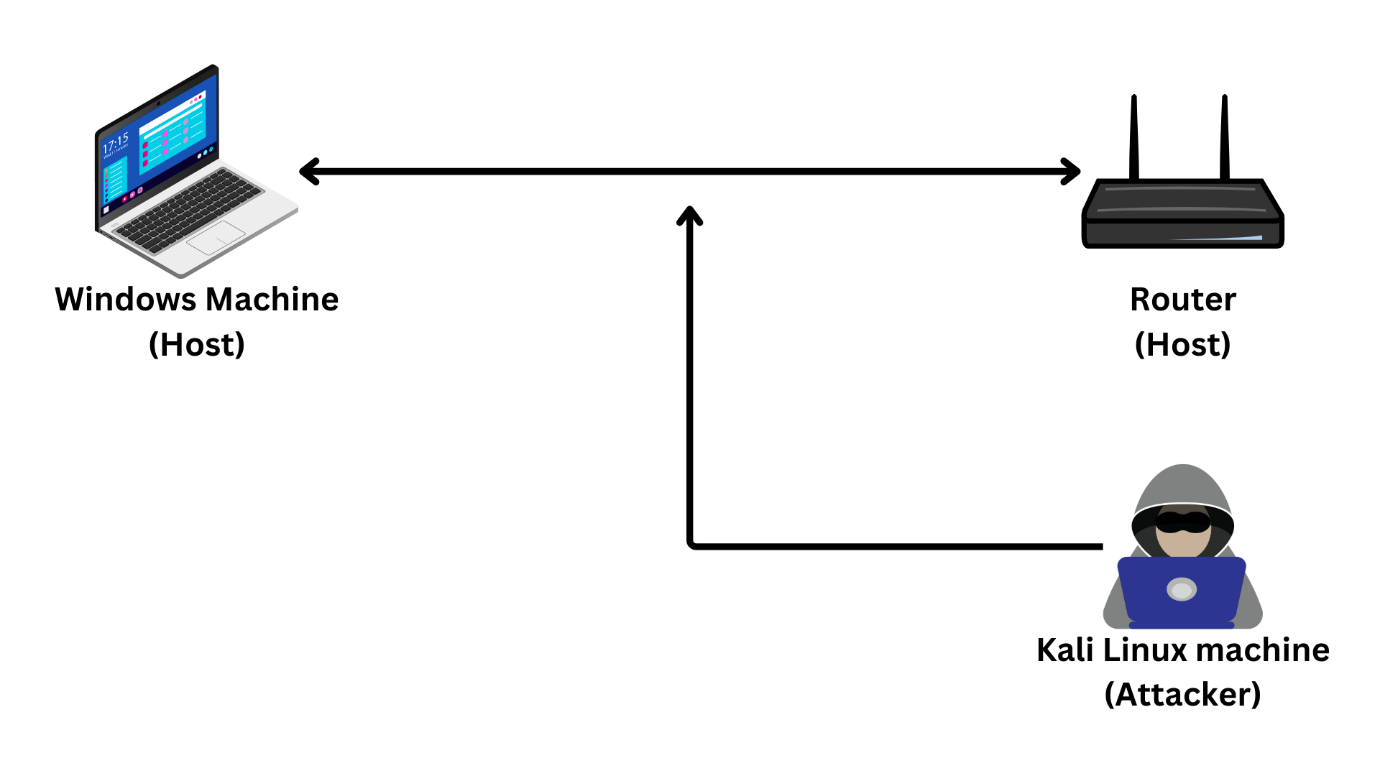
*Figure 1: Proposed learning architecture*

Figure 1 shows the proposed architecture of our learning method. ARP\_MITM and SSDP\_FLOOD datasets were used having 115 independent features and 1 target feature. First, data preprocessing was done. After data preprocessing, feature selection was done by using AfterImage Feature Extractor. By using a ML or DL algorithm, we trained our model which can then classify network packets as attack or normal.

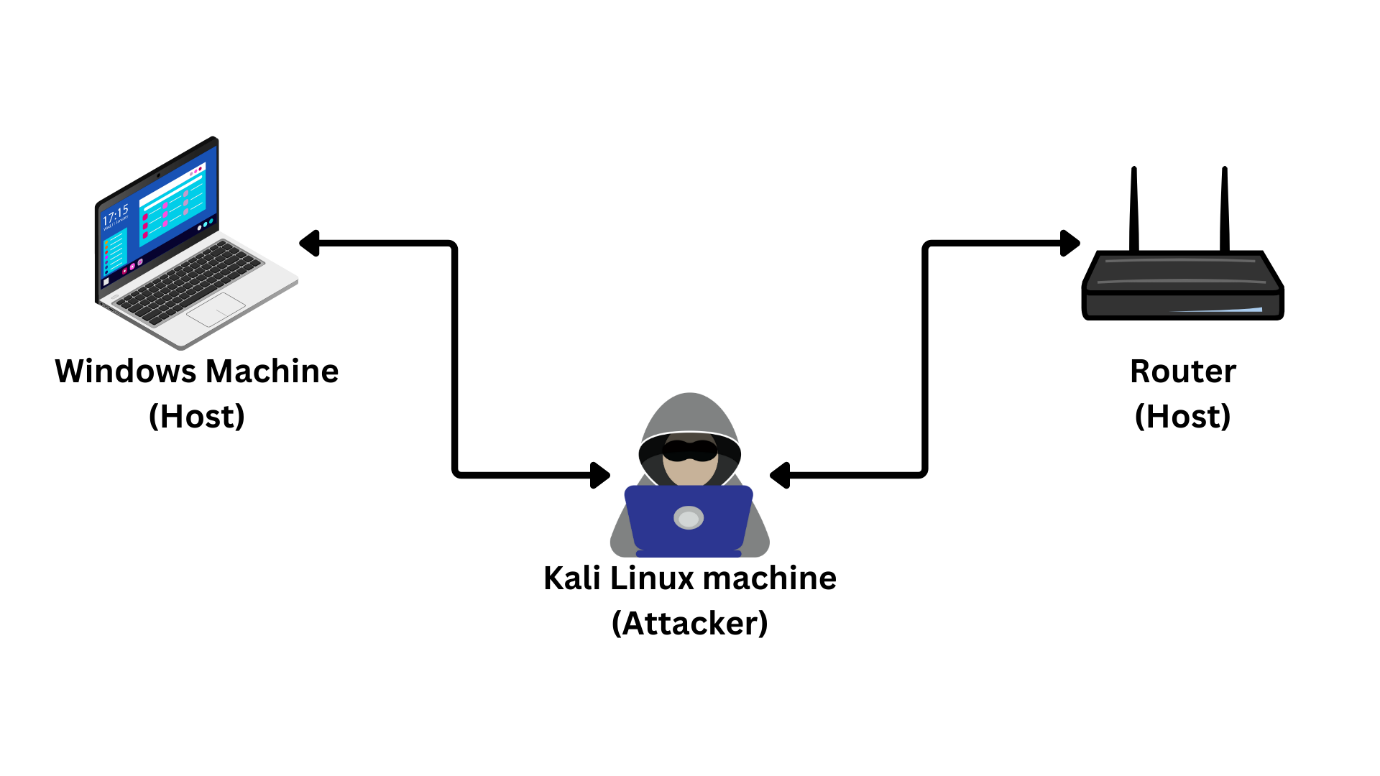


*Figure 2: Proposed IDS architecture*

Figure 2 shows the proposed architecture of our intrusion detection system in which a user will be using a website, application, or an IoT device. We will deploy our trained model on that website, application, or an IoT device and if a network intrusion happens, the deployed intrusion detection model will detect the intrusion and generate an alert.

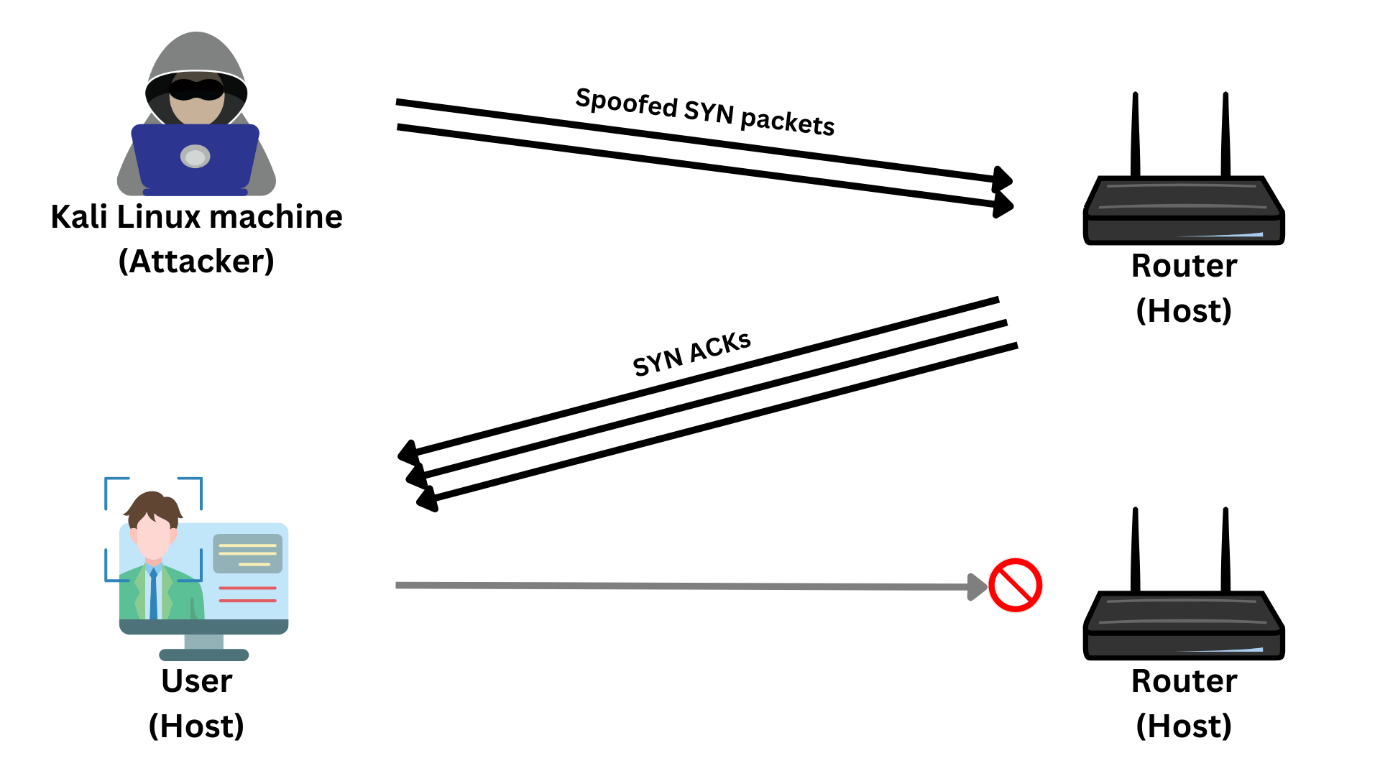


*Figure 3: MITM Attack Architecture*



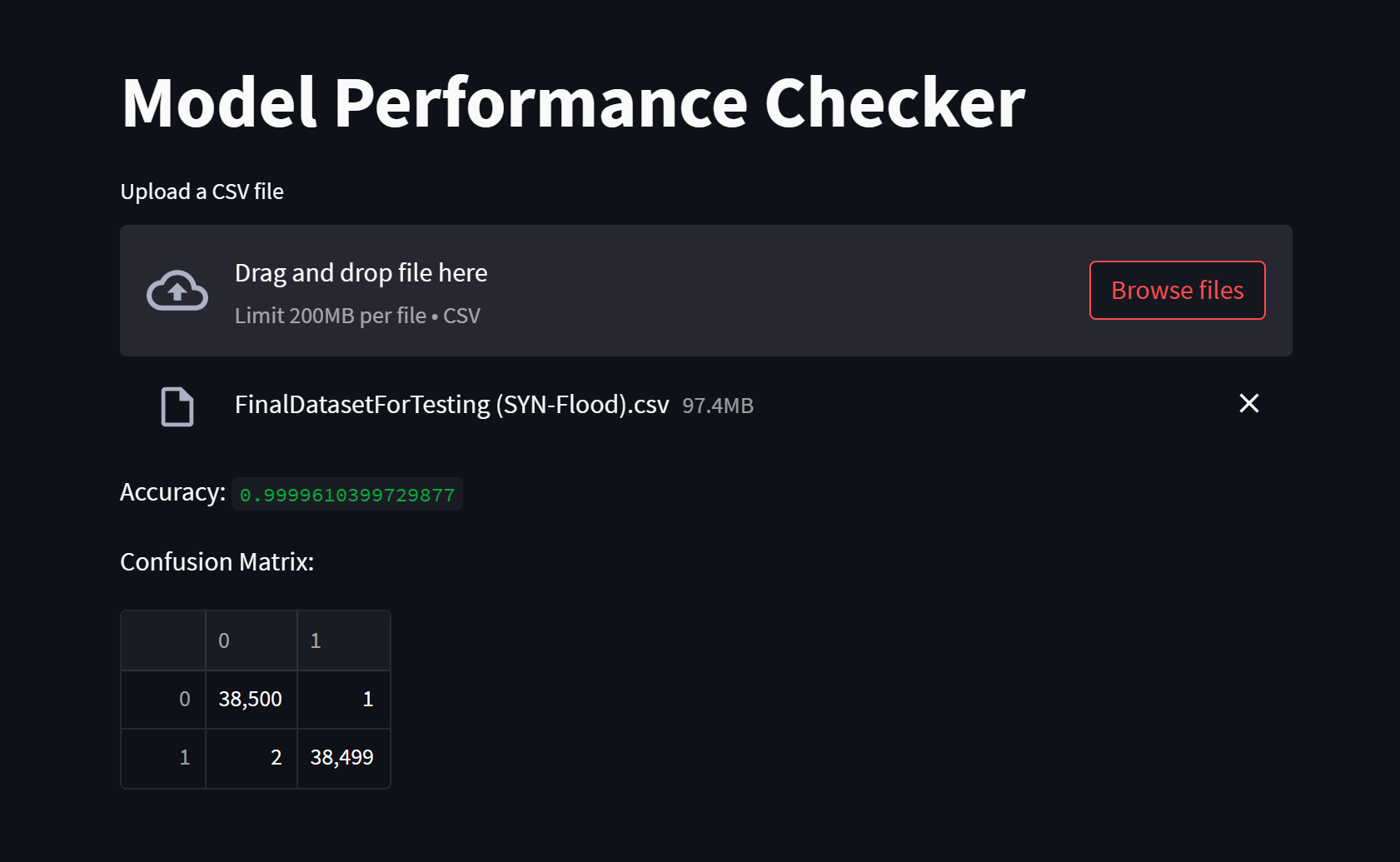
*Figure 4: MITM Attack Architecture*

In figure 3 and 4, a Windows machine and a router are interconnected, engaging in communication. The malicious attacker aims to execute a man-in-the-middle attack, assuming the role of a router for the Windows machine and a Windows machine for the router. The attacker sends forged Address Resolution Protocol (ARP) messages to the router and the host, falsely associating the attacker's MAC address with the IP address of the legitimate router. As a result, both the router and the host unknowingly send their network traffic to the attacker, believing it to be the legitimate router. This strategy enables the attacker to intercept network packets carrying desired information.



*Figure 5: SYN Flood DDoS Attack Architecture*

In figure 5, the attacker employed the hping3 tool in Kali Linux to execute a SYN Flood DDoS attack. This attack was initiated by executing a specific command in the root terminal of Kali Linux. Once the command was executed, a falsified network packet was transmitted to the router, resulting in a significant influx of SYN Acknowledgement packets being sent to the targeted user. Consequently, the excessive network traffic impeded the user's ability to utilize the services provided by the router.



*Figure 6: Front-end of our prototype*

Figure 6 illustrates the front-end of our prototype, demonstrating the dataset upload process, which includes both malicious and normal packets. Upon uploading the dataset, the back-end model conducts predictions on the packets, accurately categorizing them as normal or malicious. The front-end then presents the performance metrics and confusion matrix, providing valuable insights into the model's effectiveness.

# **CHAPTER 4**

## Project Tools

### Used Technologies

#### PYTHON:

Python is a high-level, interpreted programming language. Python is a fantastic choice for both novice and advanced programmers because of its famed simplicity, readability, and versatility. In a wide range of industries, including web development, data science, artificial intelligence, scientific computing, and education, Python is often used.

* **DATASPELL BY JETBRAINS:**

JetBrains DataSpell is an IDE for data science that includes several built-in tools, intelligent Jupyter notebooks, and interactive Python scripts. DataSpell allows you to modify and output Markdown in independent files as well as notebook cells. Jupyter Notebook doesn't have a debugger, which makes it difficult to find and fix errors in our code. Fortunately, DataSpell offers a debugger that supports both Python scripts and Jupyter notebooks. This implies that we can do actions like manipulating variables and breakpoints.

* **MACHINE LEARNING:**

Without being expressly coded, software systems may learn from experience and become more intelligent through the use of machine learning. It entails utilising a model that has been trained on a set of data to predict or decide depending on fresh incoming data.

In machine learning, a model is trained on a collection of data by being fed input and output data relating to it, and then having its internal parameters adjusted to minimise the error between the projected output and the real output. Finding a collection of parameters that can effectively generalize to novel, untested data is the aim. Using new input data and the trained model, predictions or judgements may be made.

* **DEEP LEARNING:**

Deep learning is a subfield of machine learning that is inspired by the structure and function of the brain, specifically the neural networks that make up the brain. It involves training artificial neural networks on a large dataset, allowing the network to learn and make intelligent decisions on its own.

Deep learning algorithms use multiple layers of artificial neural networks to learn and make decisions. Each layer processes the input data and passes it on to the next layer, until the final layer produces the output. The layers in between the input and output layers are called hidden layers, and the process of training a deep learning model involves adjusting the weights and biases of the connections between the neurons in these hidden layers.

* **STREAMLIT:**

Streamlit is a powerful and user-friendly Python library that allows developers to build interactive and customizable web applications for data science and machine learning projects. With Streamlit, developers can easily create and deploy web-based interfaces for data exploration, visualization, and model deployment. It simplifies the process of converting Python scripts into interactive web apps, eliminating the need for complex web development frameworks. Streamlit provides a straightforward and intuitive API that enables developers to quickly add interactive elements such as sliders, dropdowns, and plots to their applications. It also offers real-time updates, allowing users to see immediate changes as they interact with the app. Streamlit's ability to seamlessly integrate with popular data science libraries like Pandas, Matplotlib, and TensorFlow makes it a preferred choice for creating data-driven applications. Overall, Streamlit empowers developers to showcase their data science projects in an interactive and accessible manner, bridging the gap between code and end-users.

* **KALI LINUX:**

Kali Linux is a powerful and widely used operating system designed for advanced penetration testing, digital forensics, and network security assessments. Based on Debian, Kali Linux is specifically developed to provide a comprehensive and versatile platform for ethical hackers, cybersecurity professionals, and enthusiasts. It comes preloaded with a vast array of security tools and utilities, including tools for vulnerability assessment, password cracking, network scanning, wireless analysis, and data recovery. Kali Linux offers a user-friendly interface and supports various hardware platforms, making it accessible and adaptable to different environments. With its focus on security testing, Kali Linux enables professionals to identify vulnerabilities and weaknesses in systems and networks, allowing them to strengthen their security measures and protect against potential threats. It has become an indispensable tool in the cybersecurity field, facilitating robust testing, analysis, and protection of digital assets.

* **ETTERCAP:**

Ettercap is a widely used open-source network monitoring and penetration testing tool that operates on the Linux platform. It is specifically designed for network sniffing, interception, and analysis. With its powerful capabilities, Ettercap allows users to intercept and inspect network traffic in real-time, enabling them to analyze packets and gain insights into the network's structure and vulnerabilities. Ettercap supports various active and passive techniques, including ARP spoofing, DNS spoofing, and packet filtering, which can be employed to perform different types of attacks and security assessments. This tool also offers the ability to perform various man-in-the-middle attacks, facilitating the interception and manipulation of network communications between hosts. Ettercap's intuitive graphical interface and command-line options provide users with flexibility and ease of use, making it a valuable asset for network administrators, security professionals, and ethical hackers. It serves as an essential tool for assessing network security, identifying potential risks, and testing the effectiveness of security measures.

* **HPING3:**

Hping3 is a command-line network tool that provides a versatile and powerful suite of features for network testing, scanning, and manipulation. Developed for the Linux platform, hping3 is known for its flexibility and ability to craft and send custom packets to hosts, allowing users to perform a wide range of network-related tasks. With hping3, users can perform network scanning, fingerprinting, traceroute, and ping operations, among others. It supports various protocols, including TCP, UDP, ICMP, and RAW-IP, giving users fine-grained control over packet creation and modification. Hping3's advanced features include fragmentation, port scanning techniques, and the ability to craft packets with specific characteristics to simulate different types of network traffic. This tool is often utilized by network administrators, security professionals, and ethical hackers to evaluate network security, test firewall configurations, and assess the resilience of network infrastructure against potential attacks. Overall, hping3 is a valuable and flexible network tool that provides in-depth network analysis and testing capabilities.

* **WIRESHARK:**

Wireshark is a popular and powerful network protocol analyzer used for capturing and analyzing network traffic in real-time. Available for multiple platforms, including Windows, macOS, and Linux, Wireshark offers a comprehensive set of features that make it an invaluable tool for network administrators, security professionals, and researchers. With Wireshark, users can capture and inspect packets flowing through a network interface, allowing for detailed analysis of network protocols, traffic patterns, and potential security issues. It supports a wide range of protocols, from common ones like TCP/IP and HTTP to more specialized ones used in industrial control systems or VoIP. Wireshark provides a user-friendly graphical interface that displays captured packets in a readable format, allowing users to filter, search, and dissect network traffic based on various criteria. It also offers advanced features such as protocol decoding, packet reconstruction, and the ability to export captured data for further analysis. Wireshark's versatility and extensive capabilities make it an essential tool for network troubleshooting, performance optimization, network security assessments, and protocol analysis.

# **CHAPTER 5**

## Project Planning

### Project Timeline Summary

* Problem identification and background study.
* Literature review and identification of solutions.
* Implementing ML algorithms and writing methodology.
* Implementing DL algorithms and writing methodology.
* Finalizing the research paper and documentation.
* Perform attacks like MITM and DDoS and capture network packets.
* Create datasets from these network packets.
* Test our model on these datasets.
* Built a front-end where we can deploy our model and test it.

### Project Timeline Details

* *Problem identification and background study:*

Identify the issue that is affecting our society and research the solutions that have been used in the past as well as the contributions that have been made.

* *Literature Review of solutions:*

We examined many research articles to assess what contributions were made, what those contributions' limits were, and how each contribution outperformed the others.

* *Implementing ML algorithms and writing methodology:*

We learned how different machine learning algorithms operate on the backend and implemented two supervised algorithms to evaluate their effectiveness and determine which one outperforms the others. Then we documented the methodology for our research paper.

* *Implementing DL algorithms and writing methodology:*

We learned how different machine learning algorithms operate on the backend and implemented two supervised algorithms to evaluate their effectiveness and determine which one outperforms the others. Then we documented the methodology for our research paper.

* *Finalizing the research paper and documentation:*

Put everything together for research paper and start the documentation for research paper in IEEE format.

* *Perform attacks:*

Utilizing a virtualized Kali Linux environment, we will simulate and analyze man-in-the-middle and DDoS attacks on a Windows machine in order to capture network packets associated with these attacks.

* *Creation of datasets:*

Based on the network packets that have been captured, we will generate datasets comprising both malicious network packets and normal packets.

* *Testing our model:*

We will evaluate our model using the datasets we have generated. The performance assessment of our model will indicate the effectiveness of our training approach and determine its readiness for deployment in an application or website.

* *Front-end prototype:*

The front-end of our prototype, will demonstrate the dataset upload process, which includes both malicious and normal packets. Upon uploading the dataset, the back-end model conducts predictions on the packets, accurately categorizing them as normal or malicious. The front-end then presents the performance metrics and confusion matrix, providing valuable insights into the model's effectiveness.

# **CHAPTER 6**

## Project Implementation

We implemented DT, RF and Bi-LSTM on ARP\_MITM and SSDP FLOOD dataset. For RF and DT, concatenated dataset was divided into three ratios: 70%:30%, 80%:20%, and 90%:10% for training and testing respectively. Both models were trained on separate datasets as well with the same ratios.

For Bi-LSTM, concatenated dataset was used with different number of layers.

### Dataset

Kitsune is state-of-the-art cybersecurity dataset that is effective against nine various network threats, particularly on an IoT network. "Man in the Middle Attack" (MitM), Denial of service (DoS), and bot-type attacks can all be detected by it. For our model's training, we chose the SSDP FLOOD and ARP MITM datasets. The sizes of the packets and the matrix are both described in CSV files for each of them. As CSV files, they also have label files. They make a note of the packets' maliciousness or not. They have a value of ‘0-1’ Where ‘1’ means malicious and ‘0’ means normal. By looking at Kitsune's attribute information, we can determine that it has some features and relies primarily on those features.

• The packets are recorded chronologically in each row of the CSV file. Each row gives information about the channel and communication context for a given packet.

• A single window can be used to extract about "23" of the features.

• SrcIP denotes that the packets came from the source IP. SrcMAC-IP denotes packets directed from a MAC address to an IP address.

• Any protocol is used to transfer packets from source to destination (TCP or UDP).

### Methodology

#### Decision Trees (DT)

A Decision Tree is one of supervised learning algorithms that is applied to either regression or classification problems. However, it is more suitable when applied on classification problems. It is a tree-structured based classifier, where dataset's characteristics are represented by internal nodes, branches are used for process of decision making, and every last node (leaf node) gives the classification output. Decision Node and Leaf Node are the two important nodes in Decision tree. Leaf nodes are the output of decisions and do not have any more nodes connected to it, while Decision nodes are used to make decisions and they have numerous nodes connected to it. It is difficult to decide which attribute to choose as the root of the tree and internal nodes at different level if dataset contains N attributes.

So, to solve this issue of how to select attributes as root node and leaf node researchers came up with various solutions. They recommended employing standards like:

* *Entropy*

Entropy is a metric that tells us that how much information being processed is random. If the entropy is high, it becomes difficult to make any inferences from the data.

Entropy for 1 attribute is calculated using:

E(S)=

Where ‘S’ is Current state, and ‘Pi’ is Probability of an event ‘i’ of state S or Percentage of class ‘i’ in a node of state ‘S’.

Entropy for multiple attributes is calculated using:

E (T, X) =

* *Information gain*

Information gain, also known as IG, is a characteristic that evaluates how effectively a variable differentiates the training samples in accordance with their intended classification.

IG (T, X) = Entropy(T) – Entropy (T, X)

* *Gini index*

The Gini index is a cost function that is used to evaluate dataset division. It is determined by subtracting one from probabilities which are squared and added for each class.

Gini = 1 -

* *Gain Ratio*

It is the ration between Information gain and number of branches that form before splitting.

Gain Ratio =

* *Reduction in Variance*

Regression problems can be solved by this algorithm. The best split in this algorithm is done by simple variance formula. The criteria to split the population is done by selecting that has lower variance

Variance = where n represents the number of values.

* *Chi-Square*

This technique for classifying trees is among the oldest ones. It determines whether the variations between the parent node and sub-nodes are statistically significant. To calculate the sum of squares of target variable differences between observed and expected frequencies of the target variable are used.

x2 = where ‘O’ represents observed score and ‘E’ represents expected score.

#### Random Forest

We can infer from the word "forest" that there are going to be many trees there, making a forest. The Random Forest machine learning algorithm experiences the same phenomenon. In Random Forest, there are many different decision trees. Because numerous algorithms (decision trees) combine to form a single algorithm, random forest belongs to the ensemble learning category. The class that receives the most votes from each decision tree becomes the class that the random forest will forecast. A “forest” is created in Random Forest algorithm which is trained through “bootstrap aggregation” or “bagging”.

Random forest uses the ensemble approach known as bagging, which is also known as “Bootstrap Aggregation”. A sample of the dataset which is random is selected through bagging. Because of this, every model is created using the “Bootstrap Samples” provided by the original data. Additionally, there is a replacement method known as “row sampling”. “Bootstrap” is the terminology used to describe this step-in row sampling along replacement. The process of converting weak learners into strong learners with help of building successive models that have maximum accuracy possible is known as “Boosting”. For example: ADA BOOST and XG BOOST.

#### Bi-LSTM

Bi-LSTM lies in the category of RNN (Recurrent Neural Networks). Inputs and outputs are independent of each other in traditional neural networks but when there is a need to predict word or time series you have to remember previous words since the next word(output) depends on previous words (input). Similar to this, when you are watching a movie and are in the middle of it, you try to guess what will happen next in the movie using the information you have learned from viewing it. Such RNNs which remember previous inputs and information through time are called Long Short-Term Memory (LSTMs). In essence, RNNs build networks using loops to enable neural networks to take input sequences. Then, in order to reduce the complexity of the parameters being raised, independent activation is changed into dependent activation, and the same weights and biases are applied to all layers. Finally, it memorizes every previous output by using each output as the input for the subsequent layer. Since the weight and biases of these layers are same, they can be combined to form a single recurrent layer.

* *To calculate current state*

Where ht is equal to current state, ht-1 is equal to previous state and xt equals to input state.

* *To calculate output*

Where yt equals to output, why is weight at output layer.

A Bi-LSTM model contains two LSTMs, one accepts the input in forward direction and other accepts the input in backward direction. LSTM model consist of three gates:

* *Forget gate*

controls, how much data will be transferred from the memory cell from the previous step to the current memory cell.

Where ft is forget gate at t(timestep), xt is input state, ht-1 is equal to previous state, Wt is weight matrix between input gate and forget gate and bf is connection bias at time step t.

* *Input(update) gate*

It makes the decision that whether memory cell will be updated or not. Additionally, it regulates the amount of information that an existing memory cell will accept from a new memory cell.

Where it is input gate at t(timestep), xt is input state, ht-1 is equal to previous state, Wf is weight matrix of sigmoid operator between input gate and output gate and bi is connection bias at time step t

* *Output gate*

It controls the value of next hidden state

Where ot is input gate at t(timestep), xt is input state, ht-1 is equal to previous state, Wo is weight matrix of output gate and bo is connection bias at time step “t”.

### Implementation

1. *Data Gathering:*

We downloaded the kitsune dataset from Kaggle, which is state-of-the-art dataset that includes network traffic as data points with the target variable being either a benign or malicious attack. The dataset includes attacks like DoS, Man in the Middle, Injection assaults, etc. The dataset is broken up into various folders, each holding a different kind of attack. To identify MITM attacks and DOS attacks, respectively, we used Arp MITM and SSDP Flood. Compared to SSDP Flood, Arp MITM has 2504267 rows and 116 columns, while SSDP Flood has 4077266 rows and 116 columns.

1. *Model Training:*

The gathered data has been cleaned up and is prepared for training. Two supervised learning algorithms, and one neural network is used.

1. *Supervised Learning:*

We first used supervised learning algorithms to test how well our model will perform on the kitsune dataset. Decision Tree (DT) is one of the top supervised learning techniques. To determine which dataset provided greater accuracy, we trained our model on both datasets individually and together. A dataset with 6581533 rows and 116 columns was obtained by concatenating two datasets (ARP MiTM and SSDP flood). The dataset is already preprocessed for training. To balance the number of classes (0 and 1) in the train and test data during the train-test split, we used the stratify attribute on the target feature. We used three different ratios to train our model (80:20, 70:30, and 90:10). Our data was first splattered in an 80:20 ratio. 20% of the data will be used to test the model, and the remaining 80% will be used to train it. We used the default attribute settings for training our DT model. Additionally, we used Cross Validation on DT on an 80:20 ratio with a threefold k-fold value. After that, using the same parameters but without performing cross-validation on them, we applied our DT model to a 70:30 and 90:10 ratio.

After applying it to the concatenated dataset, we implemented DT on both datasets separately with the ratios 80:20, 70:30, and 90:10. To balance the number of classes (0 and 1) in the train and test data for each dataset, we used "stratify" attribute on the target feature. We did not apply cross-validation on separate datasets. The DT model attributes were default.

Random Forest (RF) is the second supervised learning algorithm that we used. In order to find which dataset provided greater accuracy, we trained our model on both datasets individually and together. A dataset with 6581533 rows and 116 columns was obtained by concatenating two datasets (ARP MiTM and SSDP flood). The dataset is already preprocessed for training. To balance the number of classes (0 and 1) in the train and test data during the train-test split, we used the stratify attribute on the target feature. We used three different ratios to train our model (80:20, 70:30, and 90:10). Our data was first splattered in an 80:20 ratio. 20% of the data will be used to test the model, and the remaining 80% will be used to train it. Our Random Forest (RF) model is trained with attribute n\_jobs = 12 and criterion = “Gini”. Additionally, we used Cross Validation on DT on an 80:20 ratio with a fourfold k-fold value. After that, using the same parameters but without performing cross-validation on them, we applied our RF model to a 70:30 and 90:10 ratio. After applying RF on the concatenated dataset, we applied RF on both datasets separately with the ratios 80:20, 70:30, and 90:10. To balance the number of classes (0 and 1) in the train and test data for each dataset, we used "stratify" attribute on the target feature. We did not apply cross-validation on separate datasets. The RF model attributes were default.

1. *Bi-LSTM:*

A dataset with 6581533 rows and 116 columns was obtained by concatenating two datasets (ARP MiTM and SSDP flood). The dataset is already preprocessed for training. To balance the number of classes (0 and 1) in the train and test data during the train-test split, we used “stratify” attribute on the target feature.

In neural network we implemented Bi-LSTM on concatenated dataset. The train-test split ratio we used was 80:20. We adjusted the layers in order to assess on how many layers our model provides greater accuracy. We tested this strategy on 2, 3, and 5 layers. Other attributes, such as units = 5, batchsize = 1028, epochs = 5, dropout = 0.2, activation = sigmoid (activation function), optimizer = Adam, loss = binary crossentropy, and returnsequences = True, validation\_split=0.1 were set.

### Results

For each of the above-mentioned ratios in supervised learning, we measured our model's Precision, F1 Score, Recall, and Accuracy metrics on both separate and concatenated datasets. We also measured AUC-score to see how well our models are trained and ROC-curve to compare DT and RF models. Precision, recall, and f1-score of all different ratios on both concatenated and separate datasets for RF and DT were 1.00. The model was not overfitted because even on unseen test dataset the accuracy was over 99%. We also applied Bi-LSTM with 2, 3, and 5 layers on the concatenated dataset to see on how many layers Bi-LSTM gives the highest accuracy. We evaluated the Bi-LSTM model by checking its accuracy, validation accuracy, train loss and test loss which can be seen in TABLE 5.

The accuracy and log-loss function of RF and DT applied to the concatenated dataset on three ratios for test data are shown in TABLE 2.

*TABLE 2: RF and DT on Concatenated dataset ON 80:20, 70:30 AND 90:10*

|  |  |  |
| --- | --- | --- |
| **DECISION TREE** | | |
| **RATIOS** | **ACCURACY** | **LOG-LOSS FUNCTION** |
| 80:20 | 99.9987% | 0.000419834 |
| 70:30 | 99.9986% | 0.004548193 |
| 90:10 | 99.9987% | 0.000419830 |
| **RANDOM FOREST** | | |
| **RATIOS** | **ACCURACY** | **LOG-LOSS FUNCTION** |
| 80:20 | 99.9997% | 7.871805e-05 |
| 70:30 | 99.9996% | 0.000104957 |
| 90:10 | 1.00 | 9.992007e-16 |

The AUC-scores of both models (RF and DT) on ARP\_MITM dataset are shown in TABLE 3 to compare the performance of both models.

*TABLE 3: AUC-scores of RF and DT for ARP\_MITM dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| **RANDOM FOREST** | | **DECISION TREE** | |
| 80:20 | 0.9999959775234588 | 80:20 | 0.9999959775234588 |
| 70:30 | 0.9999973183483589 | 70:30 | 0.9999973183483589 |
| 90:10 | 0.9999919550413943 | 90:10 | 0.9999919550413943 |

The AUC-scores of both models (RF and DT) on SSDP\_FLOOD dataset are shown in TABLE 4 to compare the performance of both models.

*TABLE 4: AUC-scores of RF and DT for SSDP\_FLOOD dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| **RANDOM FOREST** | | **DECISION TREE** | |
| 80:20 | 1.0 | 80:20 | 0.9999982634125333 |
| 70:30 | 1.0 | 70:30 | 0.9999988422736819 |
| 90:10 | 1.0 | 90:10 | 0.9999965268371295 |

*TABLE 5: Bi-LSTM on concatenated dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| **METRICS** | **2 LAYERS** | **3 LAYERS** | **5 LAYERS** |
| Accuracy | 99.5800% | 99.7300% | 99.7000% |
| Validation Accuracy | 99.8400% | 99.8400% | 99.8400% |
| Train Loss | 0.005282058 | 0.003942613 | 0.004640283 |
| Test Loss | 0.005234023 | 0.003963309 | 0.004660697 |

*TABLE 6: Comparison table between all models*

|  |  |  |  |
| --- | --- | --- | --- |
| **MODEL** | **ARP\_MITMDATASET** | **SSDP\_FLOOD DATASET** | **CONCATENATED DATASET** |
| RF | 99.9996% | 99.9997% | 99.9997% |
| DT | 99.9972% | 99.9997% | 99.9987% |
| Bi-LSTM 2-layered | - | - | 99.5800% |
| Bi-LSTM 3-layered | - | - | 99.7300% |
| Bi-LSTM 5-layered | - | - | 99.7000% |

Table 6 shows the comparison between all the models on both concatenated and separate datasets. The result shows that every model performs very accurately but Random Forest is the model which out performs every other model.

# **CHAPTER 7**

## Model Testing

Model testing is a crucial phase in the development and deployment of artificial intelligence systems. It involves evaluating the performance, accuracy, robustness, and reliability of the AI model to ensure its effectiveness and suitability for its intended purpose.

### ARP-MITM Attack and It’s Performance

We conducted an Arp Poisoning MITM attack using Kali Linux, an open-source operating system designed for ethical hacking. Kali Linux includes a built-in tool called Ettercap specifically designed for Arp poisoning attacks. In our scenario, our Windows machine served as the host and was connected to the router. Our objective was to execute the attack and position myself as the man in the middle between the Windows machine and the router.

To initiate the attack, we launched Ettercap in Kali Linux and performed a host scan. This enabled us to identify the hosts connected to the network, and we added the IP address of my Windows machine as the target. With everything set up, we initiated the MITM attack by simply pressing a button in Ettercap. As a result, our Kali machine successfully intercepted and manipulated the network traffic between the Windows machine and the router. Throughout the attack, we utilized Wireshark to capture network packets, ensuring we collected a sufficient dataset for my testing purposes. To conclude the attack, we utilized the built-in option in Ettercap to stop it.

Once we had captured the network packets, we needed to convert them into a dataset suitable for testing my AI-trained model. We obtained a dataset from Kaggle, a platform that provides datasets and resources for machine learning. They also provided Python code to create my own dataset and an Intrusion Detection System (IDS). Using their code, we transformed the captured network packets into a dataset consisting of 100 features. However, the original dataset we trained our model on had 115 features, so we needed to create 15 additional columns and populate them with zeroes to match the feature count. After creating the dataset, it contained approximately 11,000 rows, with around 6,000 rows representing normal packets and 5,000 rows representing malicious packets.

To evaluate the performance of our model, we conducted tests using different train/test ratios, specifically 20:80, 30:70, 50:50, 70:30, and 80:20. Across all these ratios, we achieved remarkable accuracy ranging from 99% to 100%.

*Table 7: Model Performance Table on ARP\_MITM*

|  |  |
| --- | --- |
| **Train/Test Ratio** | **Accuracy (in %)** |
| 10:90 | 100% |
| 20:80 | 100% |
| 30:70 | 99.987% |
| 50:50 | 100% |
| 70:30 | 100% |
| 80:20 | 100% |

### SYN-Flood DDoS Attack and It’s Performance

We conducted a SYN Flood DDoS attack using hping3, a built-in software tool in Kali Linux. In this scenario, our Kali Linux machine acted as the attacker, targeting our Windows machine as the host. To execute the attack, we entered a specific command in the root terminal of Kali Linux: 'hping3 -S 192.168.0.10 (host ipaddr) -a 192.168.0.15 (spoofing ipaddr, madeup) -p 80 (HTTP port number) –flood.' This command initiated a flood of network packets directed towards the host's IP address, overwhelming its service and causing it to become unresponsive due to the high traffic volume.

During the attack, we employed Wireshark to capture the network packets. Once we had gathered a sufficient number of packets for our test dataset, we terminated the attack by pressing 'Ctrl+C.' In just a matter of seconds, we managed to capture 55,000 packets, illustrating the significant threat this attack poses.

Following the packet capture, our next step involved converting these network packets into a suitable dataset for testing our AI-trained model. For this purpose, we obtained a dataset from Kaggle, a platform that offers datasets and resources for machine learning. Kaggle also provided Python code to create custom datasets and an Intrusion Detection System (IDS). By utilizing their code, we transformed the captured network packets into a dataset consisting of 100 features. However, the dataset we initially trained our model on contained 115 features. To address this discrepancy, we created 15 additional columns in the dataset, populating them with zeroes to match the original feature count.

Upon completing the dataset creation, we ended up with approximately 110,000 rows, with approximately 55,000 rows representing normal packets and 55,000 rows representing malicious packets. To evaluate the performance of our model, we conducted tests using different train/test ratios: 20:80, 30:70, 50:50, 70:30, and 80:20. Remarkably, across all ratios, we achieved accuracy levels ranging between 99% and 100%.

*Table 8: Model Performance Table on SYN-Flood DDoS*

|  |  |
| --- | --- |
| **Train/Test Ratio** | **Accuracy (in %)** |
| 10:90 | 99.994% |
| 20:80 | 99.995% |
| 30:70 | 99.996% |
| 50:50 | 99.994% |
| 70:30 | 99.990% |
| 80:20 | 99.990% |

# **CHAPTER 8**

## Project Prototype

### Framework

We utilized the powerful capabilities of Streamlit, an open-source Python framework, to develop a prototype of our application. Streamlit provided us with an efficient and intuitive platform for building interactive and data-driven web applications. With Streamlit, we were able to focus on the core logic and visualizations of our application, while the framework took care of the underlying web infrastructure. Its user-friendly interface enabled us to create a seamless and engaging user experience, allowing for easy data exploration and interaction with our prototype. Leveraging Streamlit's features, we successfully developed a functional and responsive application that showcased the potential of our research project.

### Front-end Working

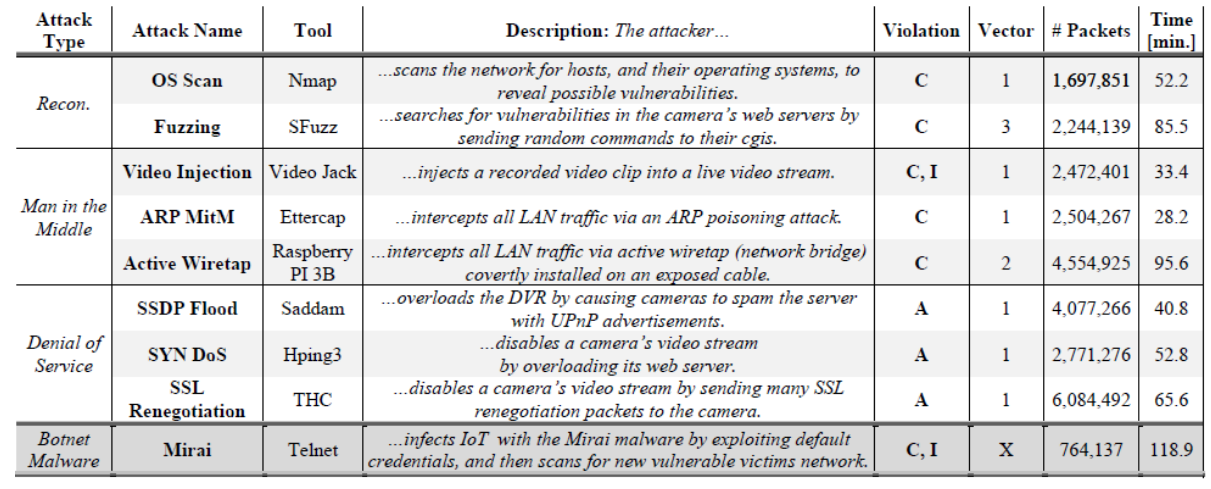
Figure 6 in chapter 3 shows the working of our prototype consist of one page. The front-end of our prototype, demonstrate the dataset upload process, which includes both malicious and normal packets. Upon uploading the dataset, the back-end model conducts predictions on the packets, accurately categorizing them as normal or malicious. The front-end then presents the performance metrics and confusion matrix, providing valuable insights into the model's effectiveness.

# **CHAPTER 9**

## Conclusion and Future Work

### Limitation

Kitsune package has variety of attacks. The cyber-attack datasets that are available are:



The whole kitsune package is of 64 GB when downloaded and unzipped. Since we didn’t have the computational resources to use all the datasets, we only used two attacks’ datasets ARP\_MITM & SSDP FLOOD. And this is the limitation of our model that it would be able to detect the attacks/anomaly, it won’t be able to identify any other attacks/anomaly other than Man In the middle and Denial of Services.

### Conclusion

This report uses machine learning and deep learning techniques to create an efficient and sustainable intrusion detection system for IoT. In machine learning, Random Forest and Decision Tree algorithms were used to detect the attacks in our intrusion detection system. Two datasets were used from kitsune which are ARP\_MiTM and SSDP\_Flood. These datasets were preprocessed containing two types of attacks (Man in the middle, denial of service). We trained our model with different ratios (80:20, 70:30, 90:10) on both datasets individually and together. To evaluate the models we used accuracy, f1-score, precision, recall, AUC score, and confusion matrix as metrics. We used the ROC curve to compare our models.

In deep learning, we used Bi-directional LSTM to detect the attacks in our intrusion detection system. Same datasets were used here. We trained our model on both datasets together with split ratio 80:20 from which we split the train data into 90:10 ratio for validation test. We used Bi-LSTM with 2,3 and 5 hidden layers having units = 5, batchsize = 1028, epochs = 5, dropout = 0.2, activation = sigmoid (activation function), optimizer = Adam, loss = binary crossentropy, returnsequences = True, and validation\_split=0.1 as parameters. All the models performs very well and had an accuracy over 99% but Random Forest outperform all the models.

We then used the best model and test it on the dataset we created by performing attacks. Our model performed really well on that dataset as well which shows that our model is not overfitted and performs really well on new data.

### Future Works

There is always a chance of improvement, following are the tips to improve the model:

* Use complete KITSUNE dataset.
* Increase the epochs and hidden layers of BI-LSTM

### References

[1] S.Kiran, R. N. K. Devisetty, N. P. Kalyan, K. Mukundini, and R. Karthi, Procedia Computer Science, vol. 171, pp. 2372–2379, 2020, doi: 10.1016/j.procs.2020.04.257.

[2] A. R. Gad, A. A. Nashat, and T. M. Barkat, IEEE Access, vol. 9, pp. 142206–142217, 2021, doi: 10.1109/ACCESS.2021.3120626.

[3] L. Ashiku and C. Dagli, Procedia Computer Science, vol. 185, pp. 239–247, 2021, doi: 10.1016/j.procs.2021.05.025.

[4] M. Anwer, G. Ahmed, A. Akhunzada, and S. Siddiqui, p. 6.

[5] Y. N. Kunang, Journal of Information Security and Applications, p. 15, 2021.

[6] A. M. Aleesa, M. Younis, A. A. Mohammed, and N. M. Sahar, vol. 16, p. 17, 2021

[7] S. M. Kasongo and Y. Sun, Computers & Security, vol. 92, p. 101752, May 2020, doi: 10.1016/j.cose.2020.101752.

[8] T. Zoppi, A. Ceccarelli, and A. Bondavalli, IEEE Access, vol. 9, pp. 90603–90615, 2021, doi: 10.1109/ACCESS.2021.3090957.

[9] Z. Wu, J. Wang, L. Hu, Z. Zhang, and H. Wu, Journal of Network and Computer Applications, vol. 164, p. 102688, Aug. 2020, doi: 10.1016/j.jnca.2020.102688.

[10] Y. Otoum and A. Nayak, J Netw Syst Manage, vol. 29, no. 3, p. 23, Jul. 2021, doi: 10.1007/s10922-021-09589-6.

### Code

Model training code:

import pandas as pd  
#%% md  
Loading ARP\_MITM dataset  
#%%  
x1 = pd.read\_csv(r'D:/FYP dataset/ARP MitM/ARP\_MitM\_dataset.csv'**,**header=None)  
x1  
#%% md  
Loading labels of ARP\_MITM  
#%%  
y1 = pd.read\_csv(r'D:/FYP dataset/ARP MitM/ARP\_MitM\_labels.csv')  
y1  
#%% md  
Loading SSDP FLOOD dataset  
#%%  
x2 = pd.read\_csv(r'D:/FYP dataset/SSDP Flood/SSDP\_Flood\_dataset.csv'**,**header=None)  
x2  
#%% md  
Loading labels of SSDP FLOOD dataset  
#%%  
y2 = pd.read\_csv(r'D:/FYP dataset/SSDP Flood/SSDP\_Flood\_labels.csv')  
y2  
#%% md  
Concatenating the features of ARP\_MITM and SSDP FLOOD datasets  
#%%  
x = pd.concat([x1**,**x2])  
x  
#%% md  
Concatenating labels of ARP\_MITM and SSDP FLOOD datasets  
#%%  
y = pd.concat([y1**,**y2])  
y  
#%% md  
Dropping the ids column  
#%%  
y.drop('Unnamed: 0'**,** inplace=True**,** axis=1)  
y  
#%% md  
Getting the number 0's and 1's in  
#%%  
print(y['x'].value\_counts())  
#%%  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report**,**confusion\_matrix  
#%% md  
Splitting into train test ratio with 80% for training and 20% for testing with stratifying 'y' meaning balancing the number of 0's and 1 in test data  
#%%  
x\_train**,** x\_test**,** y\_train**,** y\_test = train\_test\_split(x**,** y**,**train\_size = 0.8**,** test\_size = 0.2**,** random\_state = 0**,**stratify = y)  
#%%  
x\_train.shape  
#%%  
x\_test.shape  
#%%  
y\_train.shape  
#%%  
y\_test.shape  
#%% md  
Fitting the data  
#%%  
rfc = RandomForestClassifier(criterion="gini"**,**n\_jobs=12)  
rfc.fit(x\_train**,**y\_train.values.ravel())  
#%%  
import pickle  
  
with open('my\_model.pkl'**,** 'wb') as f:  
 pickle.dump(rfc**,** f)  
#%%  
y\_tr\_pred = rfc.predict(x\_train)  
y\_pred = rfc.predict(x\_test)  
#%% md  
Calculating Cross-Entropy(log\_loss) for training and testing data  
#%%  
from sklearn.metrics import log\_loss  
print("Log-Loss Function on Training Data = "**,** log\_loss(y\_train**,**y\_tr\_pred))  
print("Log-Loss Function on Testing Data = "**,** log\_loss(y\_test**,**y\_pred))  
#%% md  
Getting detailed report of all metrics for Train data and Test data  
#%%  
from sklearn import metrics  
print("\nTRAINING DATA:")  
print(classification\_report(y\_train**,** y\_tr\_pred**,** target\_names=['normal'**,** 'malicous']))  
print(metrics.accuracy\_score(y\_train**,** y\_tr\_pred))  
  
print("\nTEST DATA:")  
print(classification\_report(y\_test**,** y\_pred**,** target\_names=['normal'**,** 'malicous']))  
print(metrics.accuracy\_score(y\_test**,** y\_pred))  
#%% md  
Getting confusion matrix and value count for 0's (benign) 1's (attack) for training and test data  
#%%  
print("Number of 0's and 1's in y\_train dataset:")  
print(y\_train['x'].value\_counts())  
print("\nCONFUSION MATRIX FOR TRAINING SET: \n {}".format(confusion\_matrix(y\_train**,** y\_tr\_pred)))  
  
print("\nNumber of 0's and 1's in y\_test dataset:")  
print(y\_test['x'].value\_counts())  
print("\nCONFUSION MATRIX FOR TESTING SET: \n {}".format(confusion\_matrix(y\_test**,** y\_pred)))  
#%% md  
Visualizing Confusion Matrix  
#%%  
import seaborn as sns  
import matplotlib.pyplot as plt  
cm=confusion\_matrix(y\_test**,**y\_pred)  
f**,**ax=plt.subplots(figsize=(5**,**5))  
sns.heatmap(cm**,**annot=True**,**linewidth=0.5**,**linecolor="red"**,**fmt=".0f"**,**ax=ax)  
plt.xlabel("y\_pred")  
plt.ylabel("y\_test")  
plt.show()

Front-end Code:

import streamlit as st  
# Set the title and page layout  
st.title('Model Performance Checker')  
# st.set\_page\_config(layout='wide')  
import pandas as pd  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
import joblib  
  
# Define a function to load and preprocess the dataset  
def load\_dataset(file):  
 # Load the dataset  
 dataset = pd.read\_csv(file, header=None)  
  
 # Split the dataset into X and Y  
 X = dataset.iloc[:, :-1].values  
 Y = dataset.iloc[:, -1].values  
  
 # Normalize X using MinMaxScaler  
 scaler = MinMaxScaler()  
 X = scaler.fit\_transform(X)  
  
 # Split the dataset into 30:70 ratio for training and testing  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.7, random\_state=42, stratify=Y)  
  
 return X\_train, X\_test, Y\_train, Y\_test  
  
# Define the Streamlit web app  
def main():  
 # Add a file uploader  
 file = st.file\_uploader('Upload a CSV file', type=['csv'])  
 if file is not None:  
 # Load and preprocess the dataset  
 X\_train, X\_test, Y\_train, Y\_test = load\_dataset(file)  
  
 # Load the pre-trained model  
 model = joblib.load('my\_model.pkl')  
  
 model.fit(X\_train,Y\_train)  
  
 # Make predictions on the testing set  
 Y\_pred = model.predict(X\_test)  
  
 # Calculate the accuracy and confusion matrix  
 accuracy = accuracy\_score(Y\_test, Y\_pred)  
 matrix = confusion\_matrix(Y\_test, Y\_pred)  
  
 # Print the performance metrics  
 st.write('Accuracy:', accuracy)  
 st.write('Confusion Matrix:', matrix)  
  
# Run the Streamlit web app  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()