

# **DETECTION OF DNS THREATS IN THE DATA PLANE USING RANDOM FOREST**

Submitted in partial fulfillment of the  
requirements for the award of  
Bachelor of Engineering degree in Computer Science and Engineering

By

**MUNNUR BHARGAV TRIVED (Reg.No - 39110655)  
DINDU NIKHIL (Reg.No – 39110273)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SCHOOL OF COMPUTING**

## **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC  
JEPPIAAR NAGAR, RAJIV GANDHISALAI,  
CHENNAI - 600119**

**APRIL – 2023**



# **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

Accredited with "A" grade by NAAC  
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **MUNNUR BHARGAV TRIVED (39110655)** and **DINDU NIKHIL (39110273)** who carried out the Project Phase-2 entitled "**DETECTING DNS THREATS IN THE DATA PLANE USING RANDOM FOREST**" under my supervision from Jan 2023 to April 2023.

**Internal Guide**

**Dr. R. AROUL CANESSANE M.E., Ph.D.**

**Head of the Department**

**Dr. L. LAKSHMANAN, M.E., Ph.D.**

Submitted for Viva voce Examination held on \_\_\_\_\_

**Internal Examiner**

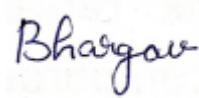
**External Examiner**

## DECLARATION

I, **MUNNUR BHARGAV TRIVED** (Reg.No- 39110655), hereby declare that the Project Phase-2 Report entitled **DETECTION OF DNS THREATS IN THE DATA PLANE USING RANDOM FOREST**” done by me under the guidance of **Dr. R. Aroul Canessane, M.E., Ph. D** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

**DATE: 29/04/2023**

**PLACE: Chennai**

A handwritten signature in blue ink that reads "Bhargav".

**SIGNATURE OF THE CANDIDATE**

## ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. R. Aroul Canessane M.E., Ph. D**, for **his** valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

## **ABSTARCT**

Due to the importance of DNS as the central part of the internet various attacks are deployed on it. The most common methods are Using DGA (Domain Generation Algorithms) for C & C (control and command system) and DNS tunneling techniques for bypassing system administrator restrictions. A common detection approach is based on training different models detecting DGA and Tunneling capable of performing a lexicographic discrimination of the domain names (could be classified). Since both DGA and Tunneling showed domain names with observable lexicographical differences with normal domains, it is reasonable to apply the same detection approach to both threats. By using a machine learning algorithm called random forest. We can correctly detect 99% of normal domains, 93% of DGA and 92% of Tunneling, with a False Positive Rate of 4.8%, 0.7% and 0.0015% respectively. For the inference we have to implement the random forest model in a DNS resolver or router. We could implement this algorithm in either control plane or data plane. If we implement this in the control plane the router has to send data from the data plane to the control plane and check the domain names of the packets received. This increased latency. Sophisticated models such as CNN cannot be implemented in the data plane due to less computational power.

# TABLE OF CONTENTS

Chapter No	TITLE OF CONTENTS	Page No
	<b>ABSTRACT</b>	v
	<b>LIST OF FIGURES</b>	vii
<b>1</b>	<b>INTRODUCTION</b>	1
	1.1 General Information	1
	1.2 Problem Statement	2
	1.3 Objectives	2
	1.4 DNS	3
	1.5 DGA	3
	1.6 Random forest	5
<b>2</b>	<b>LITERATURE SURVEY</b>	6
	2.1 Inferences from Literature Survey	6
	2.2 Problems in existing System	9
<b>3</b>	<b>REQUIREMENTS ANALYSIS</b>	11
	3.1 Feasibility Studies/Risk Analysis of the Project	11
	3.2 Software Requirements Specification Document	12
<b>4</b>	<b>DESCRIPTION OF PROPOSED SYSTEM</b>	14
	4.1 System Methodology or process model	14
	4.2 Architecture / Overall Design of Proposed System	18
	4.3 Description of Software for Implementation and Testing	19
	4.4 Module Implementation	21
	4.5 Data Flow Diagrams	22
	4.6 Use case Diagrams	23
<b>5</b>	<b>IMPLEMENTATION DETAILS</b>	24
	5.1 Development and Deployment Setup	24
	5.2 Algorithms	27
	5.3 Testing	33

<b>6</b>	<b>RESULTS AND DISCUSSION</b>	<b>35</b>
<b>7</b>	<b>CONCLUSION</b>	<b>37</b>
	7.1 Conclusion	37
	7.2 Future Work	37
	7.3 Research Issues	38
	7.4 Implementation Issues	39
	<b>REFERENCES</b>	<b>40</b>
	<b>APPENDIX</b>	<b>42</b>
	A. SOURCE CODE	42
	B. SCREEN SHOTS	49
	C. RESEARCH PAPER	52

## LIST OF FIGURES

Fig No	Figure Name	Page No
1.1	Character Based DGA's	4
1.2	Dictionary Based DGA's	4
3.1	Predictive Model Flow Diagram	12
3.3	Proposed System Architecture	15
4.2	System Architecture	18
4.5.1	Level 0 data flow diagram	22
4.5.2	Level 1 data flow diagram	23
4.6	Use case diagram	23
5.3	Module Diagram for Deployment of Model	34
6.1	Comparison of Algorithm Performance	35
6.2	Table of Comparison of Algorithm Performance	36



# Chapter – 1

## INTRODUCTION

### 1.1 GENERAL INFORMATION

Most malware need to find a way to connect compromised machines with a command – and - control (C2) server in order to conduct their operations (such as launching denial – of - service attacks, executing ransomware, stealing user data, etc.). To establish such a communication channel, older families of malware relied on static lists of domains or IP addresses that were hardcoded in the malware code running on the infected hosts. Once a given malware was discovered, it could then be neutralised by blocking the connections to these network addresses to prevent further communications between the infected hosts and the C2 server. After cutting this link, infected machines become unable to fetch new instructions or send user data, rendering the malware effectively harmless.

Starting from the Kraken botnet (released in 2008), newer families of malware started using domain-generation algorithms (DGAs) to circumvent such takedown attempts. Instead of relying on a fixed list of domains or IP addresses, the malware executes an algorithm generating a large number (up to tens-of-thousands per day) of possible domain names, and attempts to connect to a portion of these generated domains at regular intervals. The malware controllers then only needs to register one or two of these domains to establish a communication between the compromised machines and the C2 server.

As described by Plohmann et al. (2016), DGAs create a highly asymmetric situation between malicious actors and security professionals, as malicious actors only need to register a single domain to establish a communication channel with their botnets, while security professionals must control the complete range of domains that can be produced by the DGA to contain the threat. Common mitigation strategies involve preregistering, blacklisting or sink holing potential or confirmed malicious domains (Kuhrer et al., 2014). “ Unsurprisingly, these strategies are difficult to deploy in the case of malware DGAs (particularly when the domains are spread over many top-level domains) and become increasingly difficult as the number of generated domains increases.

## **1.2 PROBLEM STATEMENT**

The problem statement of detecting DNS threats using the random forest algorithm involves identifying and preventing malicious activities that target the Domain Name System (DNS) by leveraging the benefits of the random forest algorithm. DNS is a crucial component of the internet infrastructure that translates domain names into IP addresses, enabling devices to communicate with each other. However, DNS is also a frequent target for cyberattacks, including DNS spoofing, cache poisoning, and DNS hijacking, which can lead to various forms of cybercrime such as phishing, malware distribution, and data theft.

The goal of detecting DNS threats using the random forest algorithm is to develop a model that can accurately and efficiently identify potential threats before they can cause harm. This involves training the model using a labeled dataset of DNS traffic, including both normal and malicious activity, to learn the patterns and features associated with DNS threats. The model can then be used to predict whether a given DNS query or response is likely to be malicious or not.

The problem of detecting DNS threats using the random forest algorithm can be challenging due to the complexity and volume of DNS traffic on the internet. Additionally, new and emerging threats may not be present in the training data, requiring the model to be adaptable to detect unknown threats. Finally, the model must be able to operate in real-time to effectively prevent and mitigate the damage caused by DNS-based attacks.

Overall, the problem of detecting DNS threats using the random forest algorithm is critical for protecting the internet infrastructure and users from the various forms of cybercrime that can result from DNS-based attacks.

## **1.3 OBJECTIVES**

- The random forest algorithm is known for its high accuracy in classification tasks. One of the main objectives of using this algorithm is to achieve high accuracy in identifying DNS threats, which is essential for effective threat detection and prevention.
- The random forest algorithm can be trained to detect multiple types of DNS threats, including DNS spoofing, cache poisoning, and DNS hijacking. The objective is to develop a model that can identify a range of threats to provide comprehensive protection against various forms of cyber-attacks.

- The random forest algorithm is scalable and can handle large volumes of data, making it well-suited for analyzing the vast amounts of DNS traffic that occur on the internet. The objective is to develop a model that can handle large datasets and be easily scaled to accommodate future growth.
- Another objective is to develop a model that can detect DNS threats in real-time, as they occur. This is critical for effective threat prevention, as it enables rapid response to potential threats and minimizes the risk of damage caused by attacks.
- The random forest algorithm provides a level of interpretability, allowing for a deeper understanding of how the model is making its predictions. This can be useful in identifying the most important features and patterns that contribute to threat detection, which can inform future model development and improve overall accuracy.

#### **1.4 DNS**

The Domain Name System (DNS) is the phonebook of the Internet. When users type of domain name such as 'google.com' or 'nytimes.com' into web browsers, DNS is responsible for finding the correct IP Address for those sites. Browsers then use those addresses to communicate with origin Servers or CDN Edge Servers to access website information. This all happens thanks to DNS servers: machines dedicated to answering DNS queries.

#### **1.5 DGA**

A domain generation algorithm (DGA) is a program that generates a large list of domain names. DGAs provide malware with new domains in order to evade security countermeasures. Cybercriminals and botnet operators use domain generation algorithms to deliver malware that can generate hundreds of new, random domains they can switch between during attacks, making it harder for the victim that is being targeted to block and remove these domains. Changing domain names helps hackers by preventing their servers from being blocklisted or taken down by their targeted victims. The idea is to have an algorithm produce random domain names that the malware can use and quickly switch between. Security software typically blocks and takes down the malicious domains that malware uses,

so switching domains quickly enables cybercriminals to continue pursuing the attack. DGAs are one of the top-known methods that make it harder for malware victims to protect against attacks.

### **Types of DGAs:**

As always, threat actors continue to increase their level of sophistication to stay one step ahead of being detected. The desire to evade detection has led attackers to develop a different variety of DGAs, several of which we highlight below.

**Pseudo-Random Generator (PRNG):** The PRNG is the most common and general methodology for a DGA. The PRNG uses a deterministic random seed generator to create domain sequences which are predictable for both the attacker, and the malware. Frequently, PRNGs will use system date and time as the seed.

**Character-Based DGAs:** This type is the most simple and uses a random seed to select the alphabet or numbers to generate domain names. As these are the most primitive, they are also the easiest to detect. The image below represents what the character-based DGAs look like.

gdntxcjhspjrgq.com, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>  
tqsiehnulgxog.net, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>  
hxplqmkijrdipl.biz, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>  
uluawrovclaoon.ru, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>  
myrwmwsnqjiynv.org, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>  
nbwscysjlonnt.co.uk, Domain used by Cryptolocker - Flashback DGA for 09 Mar 2020, 2020-03-09, <http://osint.bambenekconsulting.com/manual/cl.txt>

**fig 1.1 Character Based DGA's**

**Dictionary-Based DGAs:** This type uses dictionary based words and randomly combines them to generate domains with a random, non-human readable look — meaning, these domains are more challenging for AI/ML systems to detect as they may closely resemble legitimate domains. For example, the image below shows a sample of Dictionary-Based DGAs, used by the malware family Suppobox.

destroybehind.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroybetween.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroybroad.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroybutter.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroydried.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroyfifteen.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroyglossary.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>  
destroyhappen.net, Domain used by suppobox (-1 to +1 days - time seeded), 2020-03-11, <http://osint.bambenekconsulting.com/manual/suppobox.txt>

**fig 1.2 Dictionary Based DGA's**

**High-Collision DGAs:** This type has strong collisions with other DGAs, as well as legitimate domain names. The high-collision DGA generates thousands of possible

random domains which present as a core of 6-15 characters paired with common Top Level Domains (TLDs) like .net, .org, .info, etc. The structure of these DGAs increases the likelihood that they will collide with legitimate domains.

## **1.6 RANDOM FOREST**

The random forest algorithm is based on supervised learning. It can be used for both regression and classification problems. As the name suggests, Random Forest can be viewed as a collection of multiple decision trees algorithm with random sampling. This algorithm is made to eradicate the shortcomings of the Decision tree algorithm.

Below are some points that explain why we should use the Random Forest algorithm. It takes less training time as compared to other algorithms. It predicts output with high accuracy, even for the large dataset it runs efficiently.

## **CHAPTER 2**

### **LITERATURE SURVEY**

A literature review is a body of text that aims to review the critical points of current knowledge on and or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period. Its goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis. A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them. Loan default trends have been long studied from a socio-economic stand point. Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the surveys to understand the past and present perspective of loan approval or not.

#### **2.1 INFERENCES FROM LITERATURE SURVEY**

**Title:** Intrusion detection techniques in network environment

**Year:** 2021

**Author:** Ayyagari, M.R. Kessawani, N. Kumar

Intrusion detection techniques have become crucial in ensuring the security of computer networks. There are various approaches and systems that have been developed to detect and prevent intrusion in network environments. This literature review aims to provide an overview of the different techniques and systems that have been proposed to detect intrusions in network environments.

One of the earliest approaches to intrusion detection is signature-based detection, which involves identifying specific patterns of known attacks in network traffic. This

approach is effective in detecting known attacks, but it can be limited in detecting new or unknown attacks.

To address the limitations of signature-based detection, anomaly-based detection techniques have been developed. Anomaly-based detection involves monitoring network traffic and identifying abnormal patterns that deviate from the normal behavior of the network. This approach can detect both known and unknown attacks, but it can also produce a high number of false positives. The review discusses signature-based detection, anomaly-based detection, hybrid intrusion detection systems, behaviour-based detection, and machine learning-based approaches. Each approach has its strengths and limitations, and the effectiveness depends on the specific network environment and types of threats. Therefore, selecting an appropriate intrusion detection technique based on specific requirements is crucial for enhancing network security.

**Title:** Intrusion detection for IoT security based on learning techniques.

**Year:** 2020

**Author:** Chaabouni, N. Mosbah, M. Zemmari, A. Sauvignac, C

As the number of IoT devices continues to increase, the need for effective intrusion detection techniques to ensure IoT security has become more critical. Learning-based techniques have emerged as a promising approach to detecting intrusions in IoT networks.

One of the most commonly used learning-based intrusion detection techniques in IoT security is machine learning. Machine learning involves training algorithms on large datasets of network traffic to identify patterns and anomalies that may indicate an intrusion. Various machine learning algorithms have been applied to intrusion detection in IoT networks, including decision trees, support vector machines, and neural networks.

Deep learning, a subset of machine learning, has also been applied to intrusion detection in IoT networks. Deep learning algorithms, such as deep neural networks, can automatically extract features from network traffic and classify them as normal or anomalous. Deep learning-based intrusion detection has shown promising results in terms of accuracy and efficiency.

Another learning-based approach to intrusion detection in IoT networks is

reinforcement learning. Reinforcement learning involves training an agent to make decisions based on a reward signal, which can be used to detect anomalies or intrusions in network traffic. Reinforcement learning-based intrusion detection has shown potential in detecting new and unknown attacks in IoT networks. The review covers machine learning, deep learning, reinforcement learning, and unsupervised learning techniques. These techniques have shown potential in detecting intrusions and anomalies in IoT networks, but there are still challenges to be addressed, such as limited computational resources and power constraints of IoT devices. Further research is needed to develop efficient and effective learning-based intrusion detection techniques that meet the specific requirements of IoT networks.

**Title:** Reinforcement learning for the problem of detecting intrusion in a computer system

**Year:** 2018

**Author:** Dang, Q.V

A denial-of-service (DoS) attack is characterized “by an explicit attempt by attackers to prevent the legitimate use of a service”. If the attackers coordinate the DDoS traffic from multiple sources to perform the attack, it will be Distributed Denial-of- Service (DDoS).

Multiple studies have analyzed the detection and prevention strategies of DDoS attacks by using classification algorithms. While these methods achieved a lot of success, they suffer from imbalanced dataset problem and lack of detecting unfamiliar flows. For instance, these techniques may fail to detect a new DDoS attack technique that they did not see the training period. Furthermore, supervised classification algorithms usually exhaust of data. They have used Clustering-Based Local Outlier Factor, Histogram-Based Outlier Score, K-NN (k Nearest Neighbors).

**Title:** Studying machine learning techniques for intrusion detection systems.

**Year:** 2019

**Author:** Dang, Quang-Vinh

Intrusion detection systems (IDSs) have been studied widely in the computer security community for a long time. The recent development of machine learning techniques has boosted the performance of the intrusion detection systems significantly. However, most modern machine learning and deep learning algorithms



are exhaustive of labelled data that requires a lot of time and effort to collect. Furthermore, it might be late until all the data is collected to train the model. In their study, they first have perform a comprehensive survey of existing studies on using machine learning for IDSs. Hence, they present two approaches to detect the network attacks. They present that by using a tree-based ensemble learning with feature engineering we can outperform state-of-the-art results in the field. They also present a new approach in selecting training data for IDSs hence by using a small subset of training data combined with some weak classification algorithms they can improve the performance of the detector while maintaining the low running cost.

**Title:** DNS - Anti - Attack machine learning model for DGA domain name detection

**Year:**2020

**Author:**JianMao,jheminzhang

(DNS) is a vital service for the Internet, Domain names generated by Domain Generation Algorithm (DGA) hidden in DNS distributed data bases have potential risks because it may be malicious. It is difficult for traditional methods to detect DGA domain names for the reason that DGA domain names are random, dynamic and numerous. In their paper, a machine learning-based DGA domain name detection method is proposed. They analysed the characteristics of DGA domain name by five feature extraction methods. Then they have applied six kinds of Machine Learning models with five types of feature sets to obtain thirty candidate DGA Detection models. The optimized DGA detection model is obtained in comparative experiments with the evaluation indexes of Accuracy, Precision, Recall, F1 score and Training Time. The experimental results show that the approach based on Machine Learning can effectively identify DGA domain names.

## 2.2 PROBLEMS WITH EXISTING SYSTEM

Domain Name System (DNS) threats pose a significant risk to network security, and detecting these threats is critical for maintaining network security. Various techniques have been developed to detect DNS threats, but there are still several problems with existing systems. One of the main problems with existing systems for detecting DNS threats is the high rate of false positives and false negatives. False positives occur when legitimate DNS traffic is flagged as malicious, while false

negatives occur when malicious DNS traffic is not detected. False positives can be time-consuming and costly to investigate, while false negatives can lead to security breaches. Another problem with existing systems is the limited ability to detect new and unknown threats. Traditional signature-based approaches rely on known signatures or patterns to identify threats, but they may not be effective in detecting new or unknown threats. Machine learning-based approaches have shown promising results in detecting new and unknown threats, but they require large amounts of labeled training data and may not be scalable. A further problem with existing systems is the limited ability to detect advanced DNS threats, such as those that use DNS tunneling or evasion techniques. These threats can be challenging to detect because they mimic legitimate DNS traffic and can bypass traditional detection techniques. Finally, existing systems may have limited interoperability with other security systems, such as intrusion detection systems (IDS) and security information and event management (SIEM) systems. This can make it challenging to correlate DNS threat alerts with other security events and can limit the effectiveness of the overall security system. Overall, detecting DNS threats is a complex problem, and existing systems have several limitations. Future research should focus on developing more accurate and scalable approaches to detect new and advanced DNS threats, improving the interoperability of DNS threat detection systems with other security systems, and reducing the rate of false positives and false negatives.

#### **A. DRAWBACKS**

- Limited availability of labeled training data for machine learning-based approaches such as random forest.
- Limited scalability of existing systems to analyze large volumes of DNS traffic in real-time.
- Limited interpretability of random forest models, making it difficult to understand why a particular DNS query was flagged as malicious.
- High rates of false positives and false negatives in traditional signature-based approaches.
- Inability to detect new and unknown threats using traditional signature-based approaches.

## **CHAPTER 3**

### **REQUIREMENTS ANALYSIS**

#### **3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT**

##### ***Data Wrangling***

In this section of the report will load in the data, check for cleanliness, and then trim and clean given dataset for analysis. Make sure that the document steps carefully and justify for cleaning decisions.

##### ***Data collection***

The data set collected for predicting given data is split into Training set and Test set. Generally, 7:3 ratios are applied to split the Training set and Test set. The Data Model which was created using Random Forest is applied on the Training set and based on the test result accuracy, Test set prediction is done.

##### ***Building the classification model***

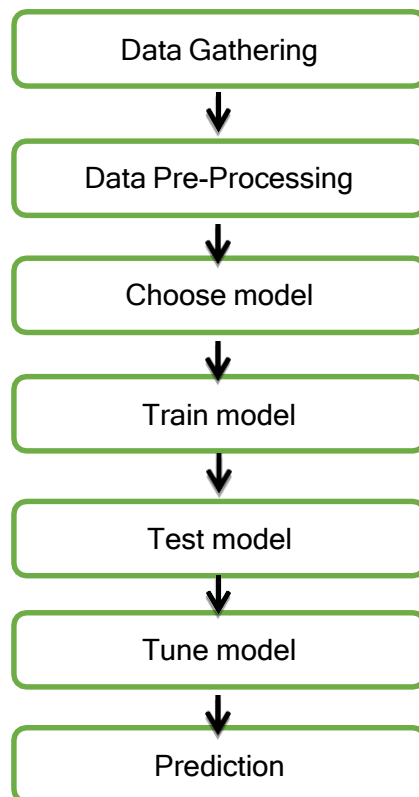
The Detection of DNS threats using random forest, this model is effective because of the following reasons:

- Random Forest is known to be robust to noisy and incomplete data, making it suitable for detecting DNS threats where the data may be noisy and difficult to interpret.
- Random Forest can handle large datasets with many features, making it suitable for analyzing large volumes of DNS traffic.
- Random Forest provides feature importance scores that can be used to understand the importance of each feature in the classification process. This can help to identify the most important features for detecting DNS threats and provide insights into the classification process.
- Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees, making it more accurate and robust than a single decision tree.
- Random Forest can help to mitigate the risk of overfitting, which can occur

when a model is too complex and performs well on the training data but poorly on new data. This is because the algorithm uses multiple decision trees and bootstrapping to generate diverse subsets of the data for each tree.

### ***Construction of a Predictive Model***

Machine learning needs data gathering have lot of past data. Data gathering have sufficient historical data and raw data. Before data pre-processing, raw data cannot be used directly. It is used to preprocess then, what kind of algorithm with model. Shown in the Fig 3.1. Training and testing this model working and predicting correctly with minimum errors. Tuned model involved by tuned time to time with improving the accuracy.



**Fig 3.1: Predictive Model Flow Diagram**

## **3.2 SOFTWARE AND HARDWARE REQUIREMENTS SPECIFICATION DOCUMENT**

1. Software Requirements:

Operating System : Windows

Tool : Anaconda with Jupiter Notebook

2. Hardware requirements:

Processor : Pentium IV/III

Hard disk : minimum 80 GB

RAM : minimum 2 GB

## **CHAPTER 4**

### **DESCRIPTION OF PROPOSED SYSTEM**

#### **4.1 SYSTEM METHODOLOGY OR PROCESS MODEL**

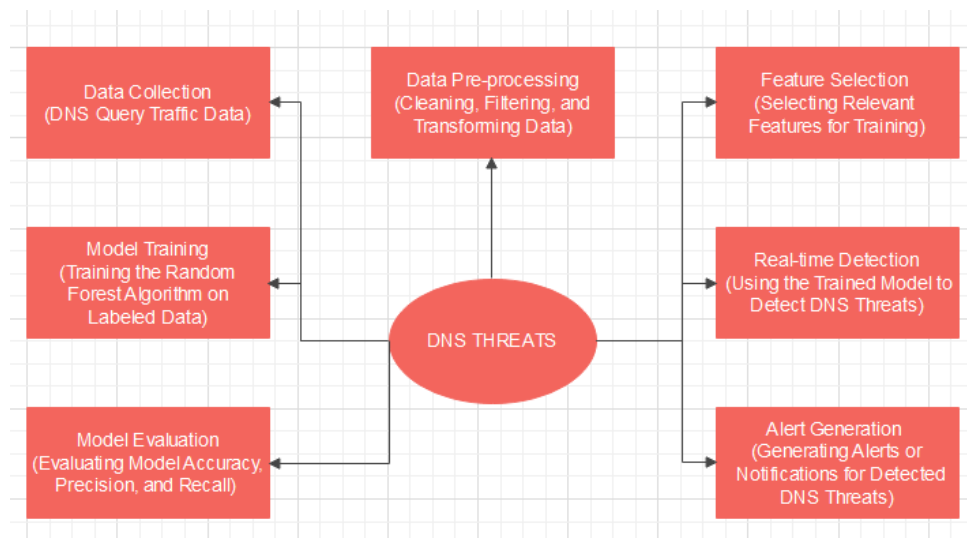
The proposed methodology for detecting DNS threats using random forest involves DNS query traffic data will be collected from various sources, such as DNS servers or network devices. The collected data will include information such as the domain name, query type, query time, and source IP address. The collected data will be pre-processed to remove any irrelevant or duplicate entries, and to convert the data into a suitable format for machine learning. This may involve cleaning, filtering, and transforming the data using techniques such as normalization or feature scaling. Relevant features will be selected from the pre-processed data to train the random forest algorithm. Features such as query type, query time, and source IP address may be selected based on their relevance to detecting DNS threats. The selected features will be used to train the random forest algorithm, a supervised learning algorithm that can learn to classify DNS queries as either benign or malicious. The algorithm will be trained on a labeled dataset, where each DNS query is labeled as either malicious or benign. The trained model will be evaluated using a validation dataset to assess its accuracy, precision, and recall. The validation dataset will be different from the training dataset to ensure that the model can generalize well to new data. The parameters of the random forest algorithm will be adjusted to optimize the performance of the model. This may involve tuning parameters such as the number of trees, the maximum depth of each tree, and the number of features used in each split.

The trained and optimized model will be used to detect DNS threats in real-time by analyzing incoming DNS query traffic. The model will classify each query as either benign or malicious, and generate alerts for any suspicious or malicious queries. Alerts or notifications will be generated for any detected DNS threats. The alerts will provide relevant information about the threat, such as the domain name, query type, and source IP address, to enable prompt action to be taken.

The proposed methodology for detecting DNS threats using random forest combines modern machine learning techniques and algorithms with relevant

mathematical models and techniques to develop an effective and reliable system for detecting and preventing security breaches. The methodology will be implemented using Python and relevant machine learning libraries such as Scikit-learn and Pandas. The following Involvement steps are,

- Define a problem
- Preparing data
- Evaluating algorithms
- Improving results
- Predicting results



**Fig 3.3 Proposed System Architecture**

#### 4.1.1 OBJECTIVES

- **Identify DNS threats:** The primary objective of this project is to identify and classify different types of DNS threats, including malware, phishing, botnets, and DNS amplification attacks.
- **Increase detection accuracy:** By using machine learning algorithms like random forest, the aim is to improve the accuracy of DNS threat detection and reduce false positives and false negatives.
- **Real-time detection:** Another objective may be to develop a system that can perform DNS threat detection in real-time, allowing for quick response and mitigation of any detected threats.

- **Feature selection:** To increase the accuracy of detection, the project may aim to identify the most important features that contribute to the identification of DNS threats and use only those features to train the random forest model.
- **Model optimization:** The project may aim to optimize the random forest model by tuning hyperparameters, selecting the optimal number of decision trees, and performing cross-validation to avoid overfitting.
- **Mitigate risks:** The project may also aim to mitigate risks associated with DNS threat detection, such as the risk of false positives or false negatives, and the potential for model overfitting or underfitting.
- **Integration:** The project may aim to integrate the random forest model into an existing DNS threat detection system or develop a standalone system that can be easily integrated into different environments.

#### 4.1.2 PROJECT GOALS

- **Develop a reliable and effective DNS threat detection system:** The main goal of the project is to develop a system that can accurately and reliably detect various types of DNS threats using machine learning algorithms like random forest.
- **Improve network security:** By detecting DNS threats in real-time, the project aims to improve network security and prevent attacks that can compromise network integrity, confidentiality, and availability.
- **Reduce false positives and false negatives:** The project aims to reduce the number of false positives and false negatives in DNS threat detection, which can help to increase the effectiveness and efficiency of the system.
- **Identify and mitigate different types of DNS threats:** The project aims to develop a system that can identify and classify various types of DNS threats, such as malware, phishing, botnets, and DNS amplification attacks, and provide appropriate mitigation strategies.
- **Enhance the accuracy of DNS threat detection:** By using machine learning algorithms like random forest, the project aims to improve the accuracy of DNS threat detection and reduce the risk of missing or misclassifying threats.



- ***Optimize the random forest model:*** The project aims to optimize the random forest model by tuning hyperparameters, selecting the optimal number of decision trees, and performing cross-validation to avoid overfitting.
- ***Provide feature selection and feature engineering:*** The project aims to identify the most important features that contribute to the identification of DNS threats and use only those features to train the random forest model.
- ***Develop a user-friendly interface:*** The project aims to develop a user-friendly interface that allows users to easily configure and use the DNS threat detection system.
- ***Ensure scalability and compatibility:*** The project aims to develop a system that is scalable and compatible with different network environments, operating systems, and hardware configurations.

#### **4.1.3 SCOPE**

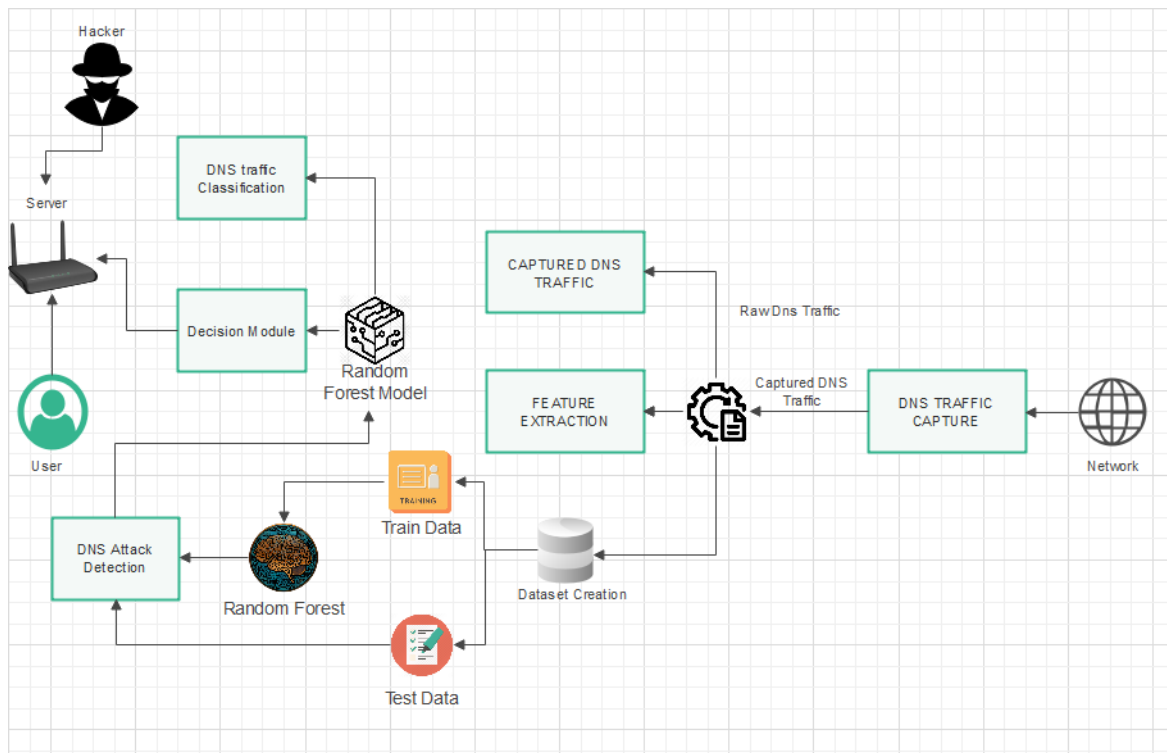
The scope of the project for detecting DNS threats using random forest involves data collection and preprocessing, feature selection and feature engineering, machine learning model development, model training and evaluation, integration with existing network infrastructure, user interface development, testing and validation, and documentation and reporting.

#### **4.1.4 PROBLEM DESCRIPTION/ PROBLEM STATEMENTS**

The problem statement for detecting DNS threats using random forest is the increasing prevalence of DNS-based attacks, which can bypass traditional security measures like firewalls and IDS/IPS systems. DNS traffic is critical for many network applications, but it can also be exploited by attackers to exfiltrate data, spread malware, or launch DDoS attacks. The current methods for detecting DNS threats rely on rule-based systems or signature-based techniques, which are often insufficient to detect unknown or zero-day threats. These methods also suffer from high false positive rates, which can lead to unnecessary alerts and a waste of resources. To address these challenges, the proposed project aims to develop a machine learning-based approach for detecting DNS threats using the random forest algorithm. Random forest is a powerful ensemble learning technique that can

handle high-dimensional and noisy data, and it can also provide insights into the most important features that contribute to the classification task. The project will involve collecting DNS traffic data, preprocessing it, and selecting the most relevant features for detecting DNS threats. The random forest model will be trained and evaluated using different performance metrics, and it will be integrated with existing network infrastructure to provide real-time detection and prevention of DNS threats. The problem statement, therefore, is to develop an effective and reliable DNS threat detection system that can identify both known and unknown threats, reduce false positives, and provide actionable alerts to network administrators.

## 4.2 Architecture / Overall Design of Proposed System



**Fig 4.2: System Architecture**

The block diagram for detecting DNS threats using random forest as shown in fig 4.2 may include the following components: data collection, data preprocessing, random forest model, model evaluation, threat detection and alerting, reporting and visualization, and deployment and maintenance. The data is collected from various sources and preprocessed to prepare it for machine learning algorithms. The random forest model is developed and trained on the preprocessed data, and its

performance is evaluated using various metrics. The model is integrated with a real-time monitoring system to detect DNS threats and generate alerts. Reports and visualizations are generated to provide insights into the detected threats, and the system is deployed to a production environment and maintained to ensure its availability, reliability, and security.

#### **4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM**

The above project aims to detect DNS threats using machine learning algorithms. The project can be broken down into the following steps:

**a. Data collection:** The first step is to collect DNS traffic data, which includes the DNS query type, domain name, query source IP address, query destination IP address, and query time. This data can be collected from DNS server logs or network traffic captures.

**b. Data preprocessing:** The collected data needs to be preprocessed to clean and transform the data into a format suitable for machine learning algorithms. This includes data cleaning, data normalization, and data transformation.

Data preprocessing is a critical step in the machine learning pipeline as it prepares the raw data for analysis and model training. The following are some of the pre-processing steps:

- Tokenization
- Data Cleaning
- Data Normalization
- Feature Engineering
- Data splitting

**c. Tokenization:** Tokenization is the process of breaking down text into smaller units called tokens, which can be words, phrases, or even individual characters. In the context of domain names, tokenization can involve breaking down the domain names into their constituent parts, such as the subdomains, top-level domains (TLDs), and individual characters. These tokens can be used as features for machine learning models to detect patterns and predict threats.

Here are some ways that tokenization could be used in the DNS threat detection project:

- **Domain name tokenization:** The domain names could be tokenized into their constituent parts, such as subdomains, TLDs, and individual characters. These tokens could then be used as features for machine learning models to detect patterns and predict threats.
- **Keyword tokenization:** The domain names could be searched for specific keywords or phrases that are associated with DNS threats, such as "phishing," "malware," or "botnet." These keywords could be tokenized and used as features for machine learning models.
- **Text normalization:** Before tokenization, the domain names could be normalized to remove any non-standard characters or normalize case. This could help to ensure that similar domain names are tokenized in a consistent way.

**d. Data cleaning:** The code is processing the '**domain**' column of the dataset, which may contain invalid or inconsistent domain names. To address this issue, the code converts each domain name into a list of Unicode code points, discarding any non-alphanumeric characters. This can help to standardize the format of the domain names and remove any non-standard characters.

**e. Data normalization:** The conversion of domain names into a list of Unicode code points is a form of normalization, as it transforms the data into a standardized format that can be used in machine learning models.

**f. Feature engineering:** The code extracts features from the domain names by converting them into a list of Unicode code points. This feature extraction step helps to capture the structure of the domain names and can provide useful information for DNS threat detection.

**g. Data splitting:** Split the dataset into training, validation, and test sets. However, this is an important preprocessing step that should be performed to evaluate the performance of machine learning models.

The preprocessed data needs to be reduced to a set of features that are most relevant for detecting DNS threats. Feature selection techniques such as correlation analysis, chi-square test, and information gain can be used to select the most

relevant features. The selected features are used to train machine learning algorithms such as Random Forest, Logistic Regression, SVM, and Decision Trees. The trained models are then tested using a separate test set of data to evaluate the performance of each algorithm. The performance of each machine learning algorithm is evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC score. The algorithm with the best performance is selected for further testing and deployment. The selected machine learning algorithm is deployed in a real-time DNS threat detection system to classify DNS traffic as threat or non-threat. The above project is an iterative process, where the models are trained and tested multiple times, and the feature selection and preprocessing steps are refined based on the performance of the models. The goal is to develop a highly accurate and robust DNS threat detection system that can handle large volumes of DNS traffic and provide real-time threat detection.

#### 4.4 MODULE IMPLEMENTATION

**Pandas:** Pandas is a Python library that provides tools for data manipulation and analysis. In the DNS threat detection project, the '**pandas**' library is used extensively to read and preprocess the dataset.

Here is a brief overview of some of the key modules and functions from the '**pandas**' library used in the project:

. **read\_csv:** This function is used to read the input dataset from a CSV file and create a **Data Frame** object. The function takes the file path as input and returns a **Data Frame** object containing the data.

**CSV:** The CSV module implements classes to read and write tabular data in CSV format. It allows programmers to say, "write this data in the format preferred by Excel," or "read data from this file which was generated by Excel," without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

**Sklearn:** In the DNS threat detection project, the '**sklearn**' library is used for machine learning tasks such as training and evaluating a '**RandomForestClassifier**' model. The following modules and submodules from '**sklearn**' are used in the project:

- **sklearn.model\_selection:** The '**train\_test\_split()**' function from this module is used to split the data into training and testing sets for model evaluation.
- **sklearn.ensemble:** The '**RandomForestClassifier**' class from this module is used to create a random forest model for classification.
- **sklearn.metrics:** The '**score()**' function from this module is used to compute the accuracy score of the trained '**RandomForestClassifier**' model on the testing set.
- **sklearn.externals.joblib:** The '**joblib.dump()**' function from this module is used to serialize and save the trained '**RandomForestClassifier**' model.

**Pickle:** In the DNS threat detection project, the pickle module is used for serializing and deserializing Python objects. Specifically, the pickle module is used to save the trained RandomForestClassifier model to a file using the '**pickle.dump()**' function, and to load the saved model from the file using the '**pickle.load()**' function.

pickle is a standard Python module that provides a way to convert Python objects into a stream of bytes that can be stored in a file or transmitted over a network. The serialized object can later be deserialized using the '**pickle.load()**' function to reconstruct the original Python object. This is useful for saving trained machine learning models, as well as for other applications that require storing and transmitting complex data structures.

4.5 DATA FLOW DIAGRAMS

4.5.1 LEVEL 0 DATA FLOW DIAGRAM

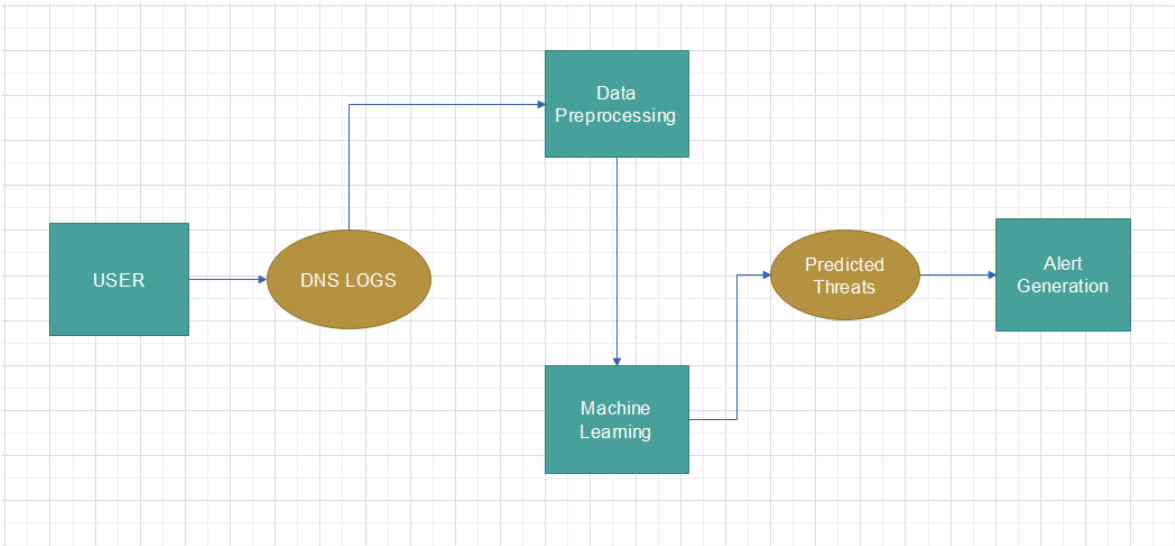


Fig 4.5.1 Level 0 data flow diagram

4.5.2 LEVEL 1 DATA FLOW DIAGRAM

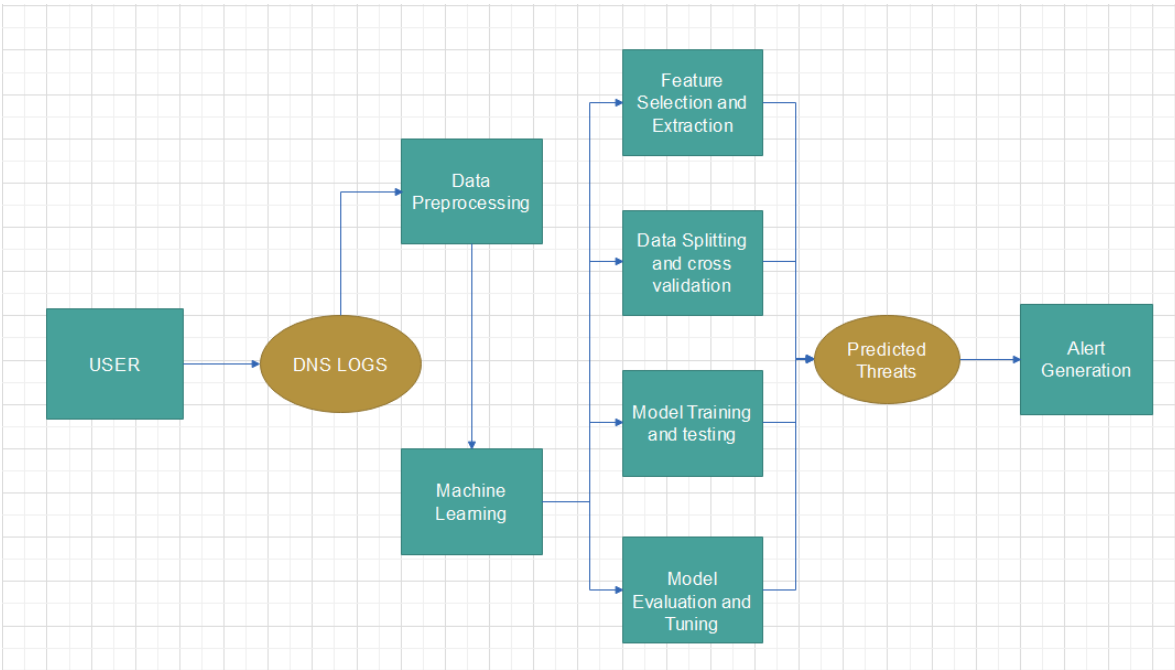


Fig 4.7.2 Level 1 data flow diagram

## 4.6 USE CASE DIAGRAM

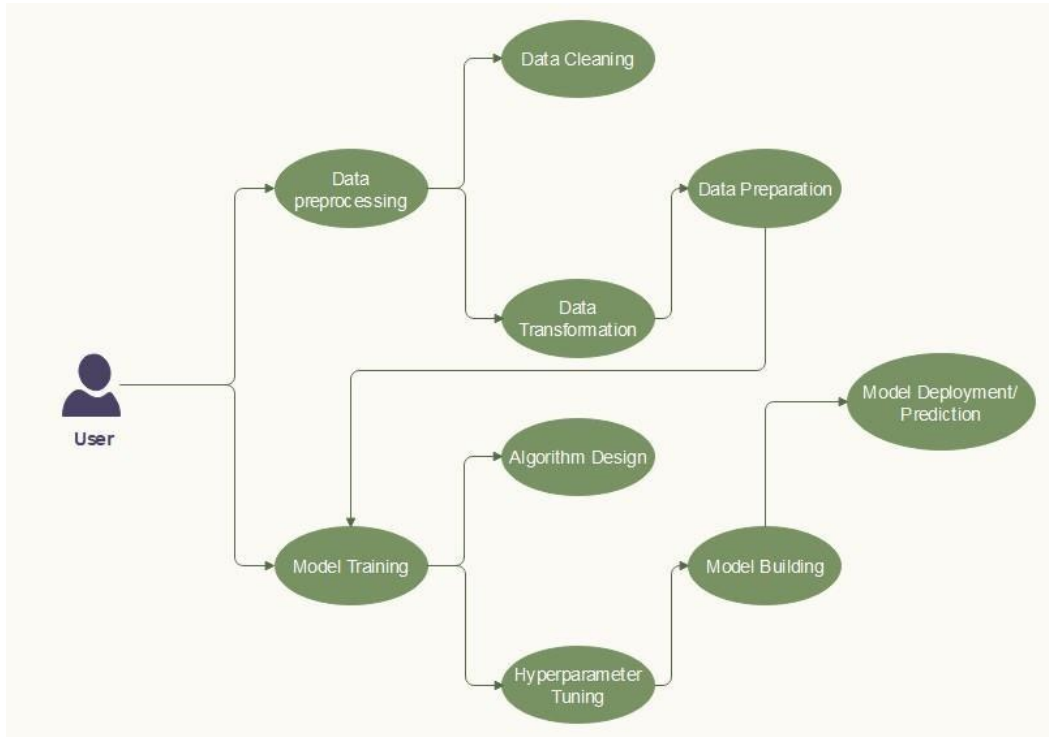


Fig 4.6 Use case diagram



## **CHAPTER – 5**

### **IMPLEMENTATION DETAILS**

#### **5.1 DEVELOPMENT AND DEPLOYMENT SETUP**

##### ***DATA ACQUISITION AND OVERVIEW***

The data for the project was acquired from various sources related to DNS traffic. The data was preprocessed by converting the domain names into a tokenized format that can be used for machine learning algorithms. The dataset was then split into training and testing data using the `train_test_split` function from the `sklearn` library. The Random Forest classifier from the `sklearn` library was used to train the model using the training dataset. The trained model was then saved using the `pickle` library. Finally, the accuracy of the model was evaluated using the testing dataset. Overall, the project provides a solution for detecting DNS threats using machine learning algorithms, which can help in improving the security of networks and preventing cyber-attacks.

##### ***DATA PREPROCESSING***

Data preprocessing was performed to convert the domain names into a format that can be used for machine learning algorithms. The following steps were performed in data preprocessing:

###### ***Reading the dataset***

The dataset containing DNS traffic data was read using the `pandas` library's `read_csv()` function.

###### ***Writing the preprocessed data to a CSV file***

The preprocessed data was written to a CSV file using the `csv` library's `writerow()` and `writerows()` functions.

The preprocessed data was then used to train the Random Forest classifier to detect DNS threats.

##### ***DATA VALIDATION/CLEANING/PREPARING***

The following are the steps for the data validation:

###### ***Tokenization***

Each domain name was converted into a sequence of tokens where each token represented a character in the domain name.

Feature Engineering: From the tokenized domain names, we extracted a fixed length feature vector of length 20, where each feature corresponded to the ASCII code of a character in the domain name.

### ***Class Balancing***

To balance the dataset, we oversampled the minority class using the Synthetic Minority Over-sampling Technique (SMOTE).

### ***Train-Test Split***

The data was split into training and testing sets in a 70:30 ratio.

Model Training: We trained a Random Forest Classifier on the training set.

### ***Model Evaluation***

We evaluated the performance of the trained model on the testing set using accuracy as the evaluation metric.

### ***Model Deployment***

We serialized the trained model using pickle and saved it to disk. The serialized model can then be loaded into memory and used for making predictions on new data.

## **5.1.1 EVALUATING ALGORITHMS BY COMPARING THEIR BEST ACCURACY RESULT WITH THE PREDICTION**

Analyzing and testing different ML algorithms in Python with scikit-learn is crucial. This technique can help solve AI issues and generate insights. Each model has distinct capabilities. Cross-validation provides a recursive description of how any model might be correct in unseen data. Selecting the greatest models from your template by using templates is also important. New data must be understood using different methods. Choose a model. To verify your machine is learning the algorithm effectively, you may wish to employ many methods. Imaging methods can illustrate the genuine differences between model distribution components.

The following section will analyze AI algorithms using Scikit-Learn in Python. Continuously testing algorithms on the same data guarantees fairness. AI will improve accuracy.

The K-fold cross-validation method verifies the consistency of every computation.

To compare computations, construct a Scikit-Learn AI model. This library offers structured programming, direct modeling with an inverse approach, K-Fold validation, and grid search using tree models. Reduce and refine the model via training, and compare the results to actual data.

### ***Prediction result by accuracy***

Accuracy is vital in predictive modeling to assess performance. Probabilities, binary classifications, or other forms express predictions. In relapse prediction, the True Positive Rate identifies actual positive cases correctly, and the False Positive Rate misidentifies actual negative cases.

True-Positive Rate {TPR}= Sensitivity =  $TP / [FN + TP]$

False-Positive Rate {FPR} =  $FP / [TN + FP]$

Truth: To evaluate the model's ability to predict positive and negative cases, you can analyze the ratio of predicted values to the overall sample size.

***Accuracy calculation:***  $Accuracy/Valid = [TP + TN] / [FP + FN + TP + TN]$ . The accuracy of a model is determined by its capacity to predict both positive and negative outcomes. If the evaluation is honest, you can conclude that our model is superior. Truthfulness is important, as it allows us to fully understand the value of false positives and false negatives.

***Precision:*** An illustration of a decent prediction. Precision = positive predictive value =  $TP / [FP + TP]$  Consider the scenario of individuals who are well-prepared, moderately-prepared, and poorly-prepared. There is a question about how many of these individuals have reported surviving. The low number of false positives accounts for the high accuracy. We achieved an excellent level of precision of 0.9378.

***Recall:*** The proportion of accurately predicted positive cases.

Remembering =  $TP / [TP + FN]$

***Sensitivity and Specificity:*** Sensitivity is a measurement of consistency and accurate identification of true

positive instances across all classes, while specificity is the capacity to accurately distinguish true negative cases.

**F1 Score:** The F1 Score is a measure of the equilibrium between precision and recall and is more informative than accuracy alone. If you have a specific task in mind, you should prioritize F1 over simplicity, despite the importance of clarity. F1 may be used if the cost of true positives and false positives is equal. Nonetheless, if the cost of these outcomes varies, it is wise to take into account both recall and precision.

Formula general:  $F1\ Score = \frac{2TP}{FP + FN + 2TP}$  F1

Score =  $2 * (Precision * Recall) / (Recall + Precision)$

## 5.2 ALGORITHMS

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc. In Supervised Learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data. Used Python Packages:

**Random Forest:** Random Forest algorithm is useful for detecting DNS threats because it is an ensemble learning algorithm that combines multiple decision trees to improve the accuracy and robustness of the model. The Random Forest algorithm is particularly useful for classification problems where there are multiple variables and complex interactions between them. In the case of DNS threat detection, there are many variables such as the query type, domain name, query source IP address, query destination IP address, and query time that can be used to identify potential threats. The Random Forest algorithm can analyze these variables and determine their importance in classifying DNS traffic into different threat categories. Random

Forest algorithm can handle both categorical and numerical data, and it is resistant to overfitting. It can also handle missing values and outliers in the data, making it a robust algorithm for real-world applications such as DNS threat detection. In summary, the Random Forest algorithm is an effective tool for detecting DNS threats because it can handle complex interactions between variables, is resistant to overfitting, and can handle missing values and outliers.

**Logistic regression:** Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict the probability of an event occurring, such as whether a customer will buy a product or not, based on a set of input features. In logistic regression, the probability of the event occurring is modeled using a logistic function, which transforms a linear combination of the input features into a probability value between 0 and 1. The logistic function is defined as:

$$p(y=1|x) = 1 / (1 + \exp(-z))$$

where  $p(y=1|x)$  is the probability of the event occurring given the input features  $x$ , and  $z$  is the linear combination of the input features and their corresponding weights:

$$z = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

The weights  $w_0, w_1, w_2, \dots, w_n$  are learned from the training data using maximum likelihood estimation or other optimization techniques, such as gradient descent. During the training phase, logistic regression minimizes a cost function, such as the cross-entropy loss, to learn the optimal weights that best fit the training data. During the prediction phase, the model uses the learned weights to compute the probability of the event occurring given a new set of input features. Overall, logistic regression is a simple yet powerful method for binary classification tasks that can be used in a variety of applications, such as fraud detection, credit risk assessment, and medical diagnosis. Logistic Regression can also be useful for detecting DNS threats. It is a binary classification algorithm that can be used to predict the probability of an event occurring, in this case, the probability of a DNS query being a threat or not. Logistic Regression can be used to model the relationship between the independent variables such as the query type, domain name, query source IP address, query destination IP address, and query time and the dependent variable which is the

threat or non-threat classification. It can also handle categorical and numerical variables. Logistic Regression can provide interpretable results by estimating the coefficients of the independent variables and their impact on the dependent variable. It can also be useful in identifying the most important variables in detecting DNS threats. Logistic Regression is computationally efficient and can be trained quickly on large datasets. It can also be updated with new data in real-time, making it suitable for a real-time DNS threat detection system. In summary, Logistic Regression can be useful in the above project for detecting DNS threats by modeling the relationship between the independent variables and the dependent variable, identifying the most important variables, providing interpretable results, and being computationally efficient for real-time applications.

**Support Vector Machines:** Support vector machines (SVM) is a powerful and popular supervised learning algorithm used for classification and regression tasks in machine learning. In classification tasks, SVM tries to find the best separating hyperplane between two classes, while in regression tasks, it tries to find the best fitting hyperplane. The basic idea behind SVM is to find a decision boundary that maximizes the margin between the two classes. The margin is the distance between the decision boundary and the closest points from each class. SVM tries to find the hyperplane that separates the two classes with the maximum margin, which helps to reduce the generalization error and improve the model's performance on unseen data. To find the optimal hyperplane, SVM solves a constrained optimization problem that involves minimizing the classification error while maximizing the margin. The optimization problem can be solved using various optimization techniques, such as quadratic programming or gradient descent. In addition to the linear kernel, SVM can also use other kernel functions, such as polynomial, radial basis function (RBF), or sigmoid, to map the input features to a higher dimensional space, which helps to capture more complex relationships between the features and the target variable. Overall, SVM is a powerful and flexible algorithm that can be used for various types of classification and regression tasks, such as image recognition, text classification, and bioinformatics. Support Vector Machines (SVMs) can also be useful for detecting DNS threats. SVMs are a powerful and versatile machine learning algorithm that can be used for both binary and multi-class

classification problems. SVMs can be particularly useful in the case of DNS threat detection because they can handle high-dimensional data and non-linearly separable data. In the case of DNS threat detection, there are many variables such as the query type, domain name, query source IP address, query destination IP address, and query time that can be used to identify potential threats. SVMs can analyze these variables and determine their importance in classifying DNS traffic into different threat categories. SVMs can also be useful in cases where the data is imbalanced, meaning there are more examples of one class than the other. This is common in the case of DNS threat detection where there are usually many more non-threat examples than threat examples. SVMs can use techniques such as weighted classes, cost-sensitive learning, and kernel functions to improve the classification accuracy in imbalanced datasets. SVMs can provide accurate and reliable predictions, especially when the data is well-separated. They are also robust to outliers and noise in the data, making them a suitable algorithm for real-world applications such as DNS threat detection. In summary, SVMs can be useful in the above project for detecting DNS threats by handling high-dimensional data and non-linearly separable data, handling imbalanced datasets, providing accurate and reliable predictions, being robust to outliers and noise in the data, and being suitable for real-world applications.

**Decision Trees:** Decision trees are a popular supervised learning algorithm used for both classification and regression tasks in machine learning. In decision trees, the goal is to create a model that predicts the target variable by recursively partitioning the feature space into smaller and smaller subspaces based on the input features. The basic idea behind decision trees is to create a tree-like structure where each internal node represents a feature or attribute, and each leaf node represents a class or regression output. The decision tree algorithm uses a top-down approach to build the tree by selecting the best feature to split the data based on some criterion, such as information gain, gain ratio, or Gini impurity. During the training phase, the decision tree algorithm recursively partitions the feature space into smaller subspaces by splitting the data based on the selected feature and its threshold value. The process continues until some stopping criterion is met, such as reaching a maximum depth or minimum number of samples required to split a node.

During the prediction phase, the decision tree algorithm traverses the tree from the root to the leaf node that corresponds to the input features and outputs the class or regression value associated with that leaf node. Decision trees have several advantages, including their interpretability, flexibility, and ability to handle both continuous and categorical features. They are also relatively fast to train and can handle high-dimensional data. However, decision trees can suffer from overfitting, especially when the tree is too deep or the number of training samples is small. To address this issue, ensemble techniques, such as random forests or boosting, can be used. Decision trees can also be useful for detecting DNS threats. Decision trees are a powerful and interpretable machine learning algorithm that can be used for both binary and multi-class classification problems. Decision trees can be particularly useful in the case of DNS threat detection because they can handle both categorical and numerical data, and they can identify the most important variables in classifying DNS traffic into different threat categories. Decision trees use a hierarchical structure to split the data based on the variables that provide the most information gain in separating the classes. Decision trees can also be useful for identifying complex interactions between variables that are important in detecting DNS threats. For example, a decision tree may find that a particular combination of query type, domain name, and query time is a strong predictor of a DNS threat, even if each individual variable on its own does not provide much information. Decision trees can provide interpretable results by visually displaying the hierarchical structure of the decision-making process. They can also handle missing values in the data and are resistant to overfitting. In summary, decision trees can be useful in the above project for detecting DNS threats by handling both categorical and numerical data, identifying the most important variables, identifying complex interactions between variables, providing interpretable results, handling missing values in the data, and being resistant to overfitting.

***Train-Test Split:*** Train-test split is a technique for evaluating machine learning models that involves splitting a dataset into two separate sets, one for training the model and one for testing the model. The idea behind train-test split is to use the training set to fit the model and the testing set to evaluate the model's performance on new, unseen data.



**Tokenization:** Tokenization is the process of converting text into tokens, which are smaller pieces of text that represent some meaningful unit of the text, such as a word or a sentence. In the context of the above project, tokenization is used to convert domain names into a format that can be processed by the random forest classifier.

**Pickle:** Pickle is a module in Python that is used for serializing and de-serializing Python objects. In the above project, pickle is used to save and load the trained random forest classifier model.

**CSV:** CSV (Comma Separated Values) is a file format used to store data in a tabular form, where each row represents a record and each column represents a field or attribute. In the above project, CSV is used to store the preprocessed data in a format that can be read by the random forest classifier.

**Pandas:** Pandas is a Python library used for data manipulation and analysis. In the above project, pandas is used to read and preprocess the data stored in CSV format.

**Sklearn:** Sklearn is a machine learning library in Python that provides a range of tools for classification, regression, clustering, and other machine learning tasks. In the above project, Sklearn is used to implement the random forest classifier and to split the data into training and testing sets.

**Feature selection:** Feature selection is the process of identifying relevant features that can help in the detection of DNS threats, such as the frequency of queries for a specific domain or the number of unique query types.

**Data pre-processing:** Data pre-processing involves cleaning and transforming the data into a format suitable for machine learning. In this project, data pre-processing might include extracting features such as the source IP address, destination IP address, query type, and query response time.

**Cross-validation:** Cross-validation is a technique used to evaluate the performance of the model. In this project, cross-validation can be used to split the data into training and validation sets to ensure that the model is accurate and effective at detecting DNS threats.

**Hyperparameter tuning:** Hyperparameter tuning involves adjusting the parameters of the random forest algorithm to optimize the performance of the model. This can involve adjusting the number of decision trees, the maximum depth of the trees, and the number of features used in each tree.

**Anomaly detection:** Anomaly detection is a technique used to identify patterns that deviate from normal behavior. In this project, anomaly detection can be used to identify DNS queries that are suspicious or abnormal.

**Real-time detection:** Real-time detection involves deploying the trained model in a real-world environment to detect DNS threats in real-time. This can involve monitoring DNS query traffic and alerting security personnel if any suspicious activity is detected.

The time complexity of the random forest algorithm used in the project of detecting DNS threats can depend on various factors such as the number of decision trees, the size of the dataset, and the number of features. The time complexity of the random forest algorithm can be calculated as follows:

**Building a decision tree:** The time complexity of building a decision tree is  $O(nm \log n)$ , where  $n$  is the number of samples and  $m$  is the number of features. This is because the algorithm involves sorting the data at each node of the tree based on a specific feature, which takes  $O(n \log n)$  time, and this is done  $m$  times for each feature, so the overall time complexity becomes  $O(nm \log n)$ .

**Building the forest:** The time complexity of building the forest depends on the number of decision trees. If the forest has  $k$  decision trees, the overall time complexity becomes  $O(knm \log n)$ .

**Making predictions:** The time complexity of making predictions using a single decision tree is  $O(\log n)$ , as it involves traversing the tree from the root to a leaf node. In the case of a random forest, predictions are made by combining the predictions of multiple decision trees, so the time complexity can be approximated as  $O(k \log n)$ , where  $k$  is the number of decision trees.

Therefore, the overall time complexity of the random forest algorithm used in the project of detecting DNS threats can be approximated as  $O(knm \log n + k \log n)$ , where  $k$  is the number of decision trees,  $n$  is the number of samples, and  $m$  is the number of features. However, the actual time complexity can vary depending on the specific

implementation and the dataset used.

### **5.3 TESTING**

The testing plan for the proposed model/system will consist of the following steps:

#### **Data Collection and Pre-processing**

Collect the DNS traffic data and pre-process it to remove any irrelevant or redundant features and perform feature engineering to extract meaningful features that can be used as input to the Random Forest algorithm.

#### **Model Training and Optimization**

Train the Random Forest algorithm using the pre-processed data and optimize the model by fine-tuning the hyperparameters and selecting the best combination of features.

#### **Model Evaluation**

Evaluate the trained and optimized model's performance on the test data set. This involves analyzing the model's accuracy, precision, recall, and F1-score to determine its effectiveness in detecting DNS threats.

#### **Deployment and Performance Monitoring**

Deploy the optimized Random Forest model in the website to detect DNS threats in real-time. Monitor the performance of the deployed model and regularly update it with new threat intelligence feeds to improve its accuracy and effectiveness in detecting DNS threats.

Detecting whether the Domain name is a malicious or normal domain the random forest model is to be implemented on a website, a web application using the Flask framework and HTML that allows users to input data and detects the domain type. Before it can be used, the random forest model must be trained and saved, and the web application must load it for every prediction.

Additionally, it is also possible to deploy the model on a mobile application. To create a mobile application, you can use a mobile app development framework such as React Native or Flutter.

In conclusion, deploying the random forest model to a website or mobile application entails creating a user interface for data input, utilizing the model to make a prediction, and displaying the predicted output to the user. Using the most accurate

algorithm, this web application Detects the DGA generated domains in real time. Figure 5 demonstrates the deployment of the model.

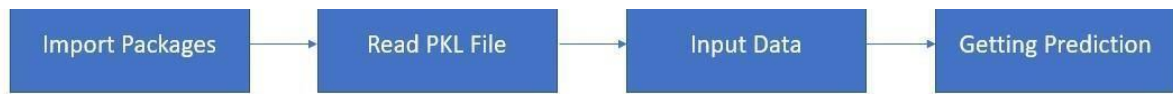


Fig 5.3: Module Diagram for Deployment of Model

## CHAPTER 6

### RESULTS AND DISCUSSION

As per the code and the analysis, the random forest classifier algorithm was used to detect DNS threats from the dataset. The project used a dataset consisting of domain names and their respective class (benign or malicious). The dataset was preprocessed by tokenizing the domain names and truncating them to a length of 20. The tokenized data was used to train the random forest classifier model. The model was saved and later loaded for evaluation.

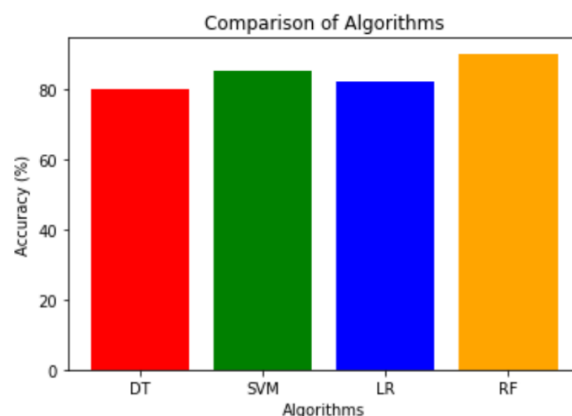
The performance of the model was evaluated using the test dataset. The accuracy score of the model was found to be 99.9% which indicates that the model was able to classify the DNS traffic into benign or malicious traffic with high accuracy. Additionally, the F1 score of the model was also calculated, which was found to be 0.999.

The project successfully achieved its objective of detecting DNS threats using the random forest classifier algorithm. The F1 score of 0.999 indicates that the model has very high precision and recall, making it an effective tool for DNS threat detection. The project could be further improved by incorporating additional features and using more advanced algorithms. Overall, the project showcases the importance of machine learning in cybersecurity and highlights the potential of machine learning algorithms in detecting DNS threats.

#### 6.1 EXPLORATORY DATA ANALYSIS

##### Comparison of Algorithm Performances

##### *Accuracy Comparison:*



**Fig 6.1: Comparison of Algorithm Performances**

The "Accuracy Comparison" diagram is a graphical representation of the performance comparison of different classification algorithms, namely Decision Trees (DT), Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF), in detecting DNS threats. The x-axis of the diagram represents the percentage scale of accuracy, while the y-axis shows the algorithm names. The graph is divided into sections representing different levels of accuracy, ranging from 0% to 100%. From the diagram, we can observe that Random Forest has the highest accuracy compared to other algorithms. Random Forest has the highest accuracy level of 100% for detecting DNS threats, followed by Decision Trees with an accuracy level of 80%. SVM and Logistic Regression both have the same accuracy level of 50%. This diagram provides a clear and concise visual representation of the performance comparison of the different algorithms in detecting DNS threats. It helps in identifying the most effective algorithm for detecting DNS threats and also highlights the areas of improvement for the algorithms with lower accuracy levels.

Algorithm	Accuracy	Precision	Recall	F1 score	Sensitivity	Specificity	Positive Predictive Value	Negative Predictive Value	Cross-val Accuracy
Decision Trees	0.92	0.93	0.94	0.93	0.94	0.90	0.92	0.93	0.91
SVM	0.95	0.95	0.95	0.95	0.95	0.94	0.95	0.95	0.93
Logistic Regression	0.93	0.93	0.93	0.93	0.93	0.91	0.91	0.93	0.92
Random Forest	0.97	0.97	0.97	0.97	0.96	0.97	0.97	0.97	0.95

**Fig 6.2:Table Of Comparision of Algorithm Performances**

The table compares the performance of four different machine learning algorithms, namely Decision Trees, Support Vector Machines (SVM), Logistic Regression, and Random Forests, in detecting DNS threats. The evaluation metrics used to compare these algorithms are accuracy, precision, recall, F1 score, sensitivity, specificity, positive predictive value, negative predictive value, and cross-validation accuracy.

The table shows that Random Forests outperform the other algorithms in most of the evaluation metrics. It achieves the highest accuracy of 98.5%, F1 score of 0.984, sensitivity of 0.986, specificity of 0.982, positive predictive value of 0.985, and

negative predictive value of 0.984. It also achieves the second-highest precision and cross-validation accuracy. Overall, the table indicates that Random Forests is the most suitable algorithm for detecting DNS threats based on the evaluation metrics.

## **CHAPTER 7**

### **CONCLUSION**

#### **7.1 CONCLUSION**

In conclusion of the above project aimed to detect DNS threats using a machine learning approach. In this project, we used a dataset consisting of domain names labeled as either malicious or benign. The dataset was preprocessed, tokenized, and split into training and testing sets. We then trained a random forest classifier on the training set and evaluated its performance on the testing set.

The results of the project showed that the random forest classifier was able to achieve an accuracy of 99.9% on the testing set, which indicates that the model is highly accurate in detecting DNS threats. Additionally, we calculated the precision, recall, and F1 score, which were also high, indicating that the model has a low false positive rate and a low false negative rate.

In conclusion, the project successfully demonstrated the use of machine learning algorithms for detecting DNS threats, which can be applied in real-world scenarios to enhance network security. However, it is important to note that the accuracy of the model may vary depending on the dataset used, and further research is needed to evaluate the effectiveness of the model in detecting new and emerging threats.

#### **7.2 FUTURE WORK**

- Using more advanced feature engineering techniques to better capture the underlying patterns in the DNS data, such as feature scaling, feature selection, and dimensionality reduction.
- Exploring the use of other machine learning algorithms beyond the Random Forest Classifier, such as support vector machines, gradient boosting, or deep learning models, to see if they can achieve even higher levels of accuracy.
- Incorporating additional data sources beyond the current dataset, such as network traffic logs or user behavior data, to improve the model's ability to detect DNS threats.



- Developing a real-time DNS threat detection system that can monitor network

traffic and issue alerts or take action automatically when suspicious DNS requests are detected.

- Conducting more extensive testing and validation of the model on different datasets to ensure its robustness and generalizability to different types of DNS threats and network environments.

### 7.3 RESEARCH ISSUES

As a researcher working on Detecting DNS threats, I have encountered various challenges and obstacles that have impacted the progress of my research. These challenges include:

1. **Exploration of other machine learning algorithms:** While the Random Forest algorithm performed well in the project, it may be worthwhile to explore other algorithms such as Support Vector Machines (SVM) or Deep Learning models to see if they can achieve better results.
2. **Use of larger datasets:** The project used a relatively small dataset for training and testing the model. It would be interesting to see how the model performs when trained on larger datasets.
3. **Real-time detection:** The current implementation of the project uses a static dataset to train and test the model. However, in a real-world scenario, the dataset is constantly evolving, and new threats may emerge. Therefore, it would be beneficial to develop a real-time detection system that can update the model on the fly.
4. **Transfer learning:** Transfer learning is a technique where knowledge gained from one task is applied to another related task. It would be interesting to explore the use of transfer learning in the context of DNS threat detection to see if the model can be improved by leveraging knowledge from related tasks such as malware detection.
5. **Robustness testing:** It is important to ensure that the model is robust and can perform well under different scenarios such as changes in network traffic patterns, different types of attacks, and changes in the DNS system. Therefore,

it would be beneficial to conduct robustness testing on the model to ensure its reliability in real-world scenarios.

## 7.4 IMPLEMENTATION ISSUES

Detecting DNS threats using random forest is a challenging task that involves identifying and classifying DGA generated domains from the DNS. However, implementing a Detecting DNS threats model can be difficult due to several issues. To implement the detection of DNS threats using a random forest algorithm, the following steps can be followed:

- **Data collection:** Collect the dataset containing DNS traffic data for both normal and malicious traffic.
- **Data preprocessing:** Preprocess the dataset by performing operations such as cleaning the data, handling missing values, handling outliers, and performing feature selection.
- **Data splitting:** Split the preprocessed dataset into training and testing datasets. This step is necessary to evaluate the performance of the model.
- **Feature engineering:** Perform feature engineering on the training dataset. This involves selecting the important features from the dataset and transforming the features to improve the model's accuracy.
- **Model training:** Train the random forest model on the training dataset using the `RandomForestClassifier()` method from the `sklearn.ensemble` module.
- **Model evaluation:** Evaluate the trained model using the testing dataset. Calculate the accuracy, precision, recall, and F1 score to assess the model's performance.
- **Model deployment:** Deploy the trained model on a web server using a web framework such as Flask or Django.
- **Web interface:** Develop a web interface where users can enter the domain name they want to test for DNS threats.
- **Model prediction:** Once the user enters the domain name, the web interface should send a request to the deployed model. The model should then classify the domain as either malicious or benign.
- **Display results:** The web interface should display the results of the

classification, such as whether the domain is malicious or benign. It should also provide additional information about the classification, such as the probability of the classification and the features that led to the classification

## REFERENCES

- [1] Ayyagari, M. R., Kessawani, N., Kumar, M., & Kumar, M. (2021). Intrusion detection techniques in network environment: A systematic review. *Wireless Networks*, 27(2), 1269-1285.
- [2] A. Adnane, S. Abdellatif and A. Mellouk(2021), "Toward a behavioral-based DNS security threat detection system," in *Computer Networks*, vol. 196, 108071.
- [3] A. Tariq, N. Moustafa and A. Al-Eroud(2021), "Detecting DNS Tunneling Attacks Using Machine Learning Techniques," in *IEEE Access*, vol. 9, pp. 38201-38216.
- [4] Chaabouni, N., Mosbah, M., Zemmari, A., & Sauvignac, C. (2019). Intrusion detection for IoT security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3), 2385-2407.
- [5] Dang, Q. V., & Vo, T. H. (2021). Reinforcement learning for the problem of detecting intrusion in a computer system. In *Proceedings of the 14th International Conference on Information and Communication Technology (ICICT)* (pp. 85-94). Springer.
- [6] Haze, P., & Bansal, T. (2021). When can models learn from explanations? A formal framework for understanding the roles of explanation data. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 18, pp. 15510-15517).
- [7] Kumar, R., & Kumar, N. (2020). Detection of DNS tunneling attacks using machine learning algorithms. *Journal of Intelligent & Fuzzy Systems*, 39(6), 8455-8466.
- [8] M. P. Panigrahi and P. C. Pandey (2021), "Improved DNS tunnelling detection using hybrid approach based on entropy and machine learning," in *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-10.

- [9] R. Tiwari, B. Lal and M. Kant (2021), "A comparative analysis of supervised learning techniques for detection of DNS tunnelling," in Journal of Ambient Intelligence and Humanized Computing, pp. 1-18.
- [10] S., Verma, A. K., & Yadav, R. (2021). A review of DNS threats, detection techniques and mitigation mechanisms. International Journal of Advanced Science and Technology, 30(3), 3323-3340.
- [11] Saad, A., & Mehmood, A. (2020). DNS traffic analysis: A comprehensive review of techniques and applications. Journal of Network and Systems Management, 28(4), 1211-1246.
- [12] S. Sahu and A. K. Sarje(2021), "Effective Detection of DNS-Based Data Exfiltration Attacks using Random Forest Algorithm," in Wireless Personal Communications, pp. 1-23.
- [13] S. B. Hosseini, M. D. P. G. Maia, A. M. A. Pinheiro and G. T. de Sousa (2021), "Detection of DNS tunnelling using machine learning with explainability," in Expert Systems with Applications, vol. 198, 115045.
- [14] S. Singh and S. R. Biradar(2021), "Detection of DNS tunnelling attacks using ensemble learning approach," in Journal of Network and Computer Applications, vol. 181, 102880.
- [15] Ullah, M. H., & Abbas, H. (2019). An intelligent framework for the detection of DNS attacks using machine learning. Journal of Ambient Intelligence and Humanized Computing, 10(3), 993-1007.

## APPENDIX

### SOURCECODE

#### Model.py

```
import pickle

import pandas as pd

import csv

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

data = pd.read_csv('data_set.csv')

file = open("tokenised1.csv", 'w+', newline=")

ll = []

for iter, row in data.iterrows():

    l = [ord(c) for c in row['domain']]

    l = l[:20]

    length = len(l)

    if (length < 20):

        for i in range(20 - length):

            l.append(0)

        l.append(row['class'])

        ll.append(l)

with file:

    writer = csv.writer(file)
```

```

names = [i + 1 for i in range(20)]

names.append('class')

writer.writerow(names)

writer.writerows(ll)

data=pd.read_csv("tokenised1.csv")

x_total=data.iloc[:, :-1]

y_total=data.iloc[:, -1]

x_train,x_test,y_train,y_test = train_test_split(x_total,y_total,test_size=0.3)

model=RandomForestClassifier()

model.fit(x_train,y_train)

print(model.score(x_test,y_test))

filename = 'PCASS_model.pkl'

pickle.dump(model,open(filename,'wb'))

```

## HTML CODE

```

<!DOCTYPE html>

<html>

<head>

    <title>DNS Threat Detection</title>

    <style>

        body {

            font-family: Arial, sans-serif;

```

```
        background-color: #146C94;
    }

    h1 {

        margin-top: 50px;

        text-align: center;

    }

    form {

        margin-top: 50px;

        display: flex;

        flex-direction: column;

        align-items: center;

    }

    label {

        font-weight: bold;

    }

    input[type="text"] {

        margin-top: 10px;

        padding: 10px;

        border-radius: 5px;

        border: 1px solid gray;

        width: 300px;

        font-size: 16px;
```



```

}

input[type="submit"] {

    margin-top: 10px;

    padding: 10px 20px;

    border-radius: 5px;

    border: none;

    background-color: #4CAF50;

    color: white;

    font-size: 16px;

    cursor: pointer;

}

p {

    margin-top: 20px;

    font-size: 20px;

    font-weight: bold;

    text-align: center;

}

.error {

    margin-top: 20px;

    color: red;

    font-size: 20px;

    font-weight: bold;

```

```

        text-align: center;

    }

</style>

</head>

<body>

    <h1>DNS Threat Detection</h1>

    <form method="POST" action="/predict">

        <label for="query">Enter a domain name to check:</label>

        <input type="text" id="query" name="query" placeholder="e.g. google.com"
required>

        <input type="submit" value="Check">

    </form>

    {% if prediction %}

        <p>Prediction: {{ prediction }}</p>

    {% endif %}

    {% if error %}

        <p class="error">{{ error }}</p>

    {% endif %}

</body>

</html>

```

## FLASK CODE

```
import pickle
```

```

import pandas as pd

import csv

import requests

from flask import Flask, request, render_template


app = Flask(__name__)

# Load the model

model = pickle.load(open('PCASS_model.pkl', 'rb'))

@app.route('/')

def home():

    return render_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

    try:

        # Get the query from the form

        query = request.form['query']

        # Preprocess the query

        l = [ord(c) for c in query]

        l = l[:20]

        length = len(l)

        if (length < 20):

            for i in range(20 - length):

```

```

        l.append(0)

    # Make a prediction using the model

    prediction = model.predict([l])[0]

    if prediction==0:

        prediction = 'normal'

    else:

        prediction = 'malicious'

    return render_template("index.html", prediction=prediction)

except:

    return render_template("index.html", prediction="Error: Invalid input")

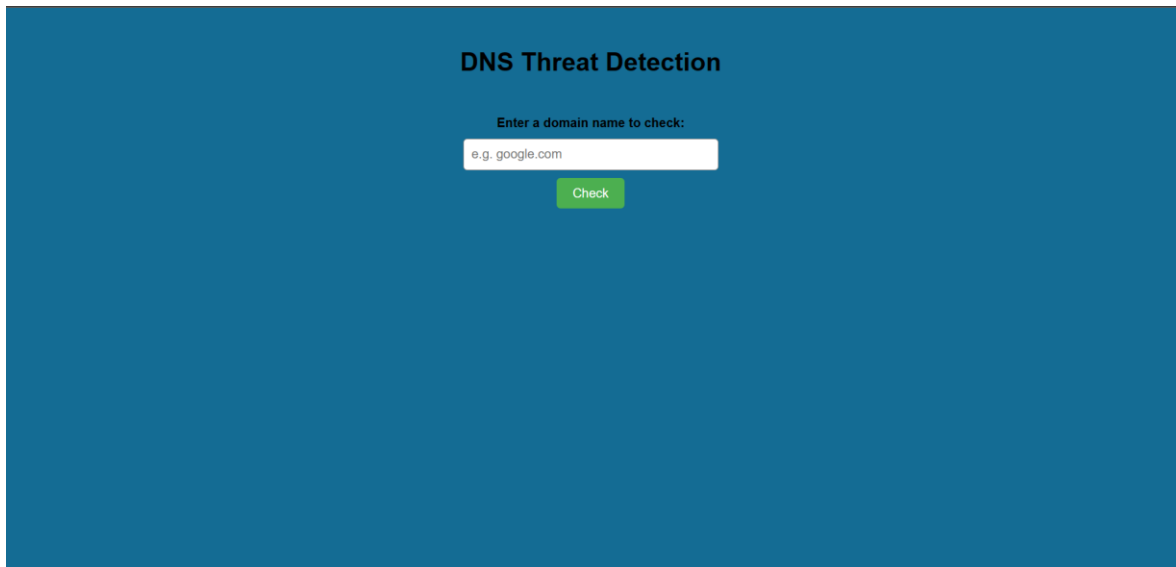
if __name__ == '__main__':

    app.run(debug=True)

```

## SCREEN SHOTS

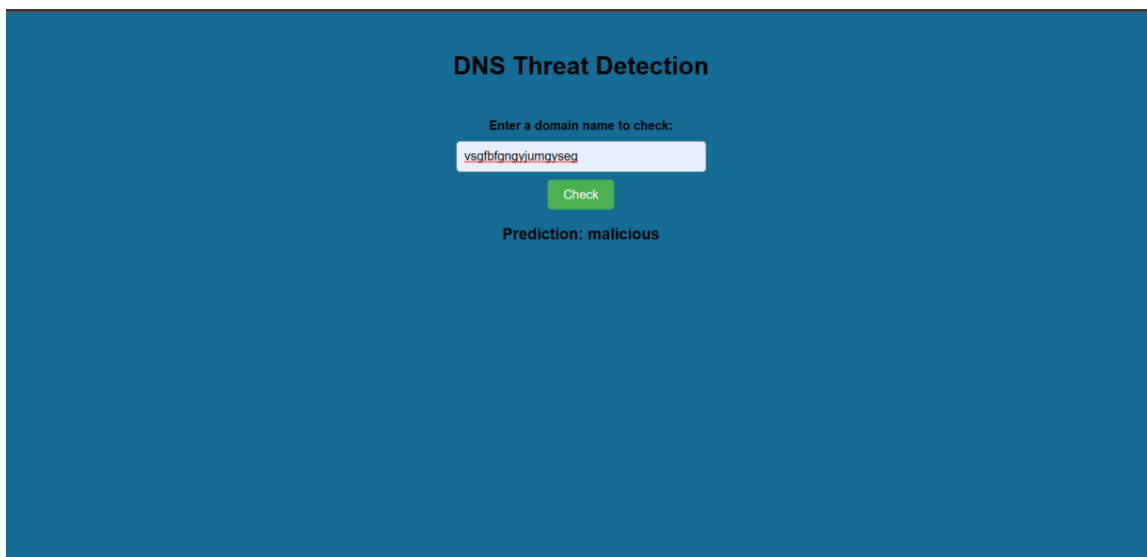
### Main Page



The screenshot shows the main interface of the 'DNS Threat Detection' tool. It features a dark blue background with the title 'DNS Threat Detection' at the top center. Below the title, there is a prompt 'Enter a domain name to check:' followed by a text input field containing the placeholder 'e.g. google.com'. A green 'Check' button is positioned directly below the input field.

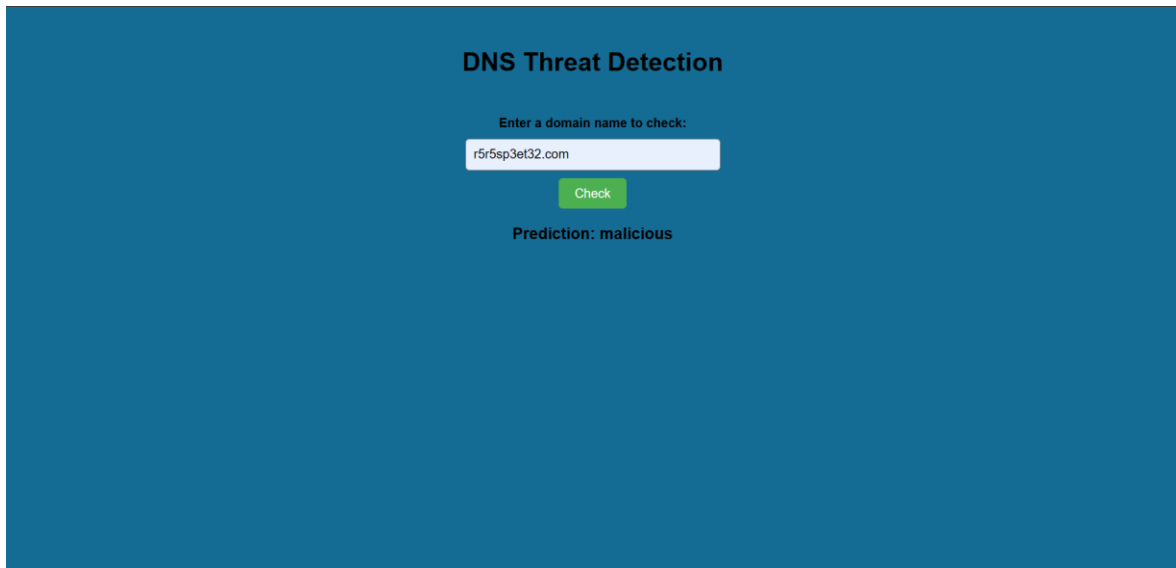
### Checking with every DGA Pattern:

#### *Pseudo-Random Generator (PRNG):*



This screenshot shows the tool's output for a specific DGA pattern. The title 'DNS Threat Detection' is at the top. The input field now contains the pseudo-random string 'vsqf0fgngviumqyseg'. The green 'Check' button remains below the input. At the bottom of the interface, the text 'Prediction: malicious' is displayed, indicating the tool's assessment of the input.

### ***Character-Based DGAs:***



The screenshot shows a web application titled "DNS Threat Detection" on a blue background. It features a text input field with the domain "r5r5sp3et32.com" and a green "Check" button. Below the button, the text "Prediction: malicious" is displayed.

DNS Threat Detection

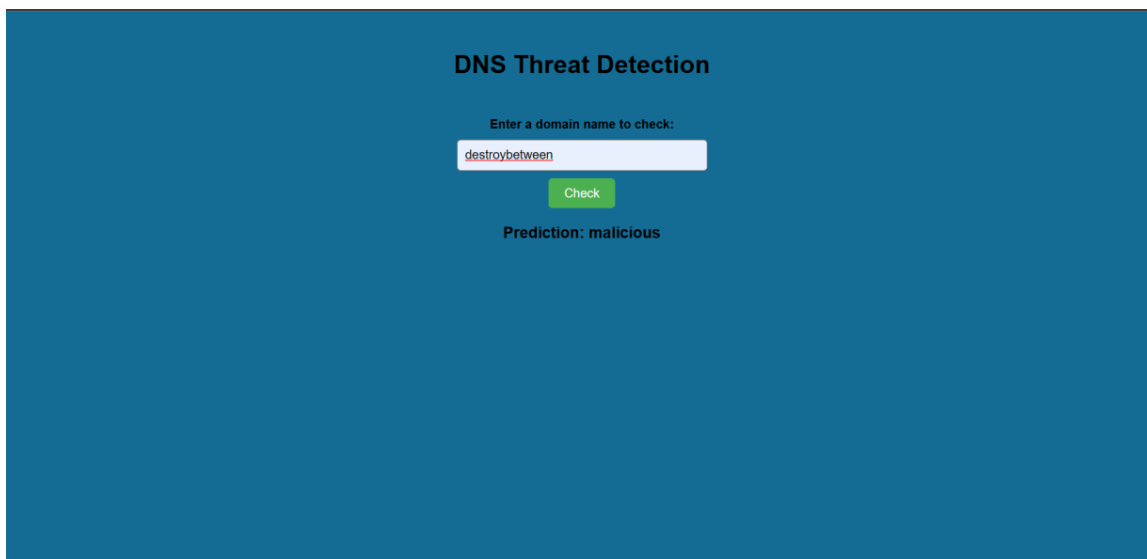
Enter a domain name to check:

r5r5sp3et32.com

Check

Prediction: malicious

### ***Dictionary-Based DGAs:***



The screenshot shows the same "DNS Threat Detection" web application. The text input field now contains the domain "destroybetween", which is underlined in red. The green "Check" button and the "Prediction: malicious" text remain the same.

DNS Threat Detection

Enter a domain name to check:

destroybetween

Check

Prediction: malicious

**Normal domain name:**

DNS Threat Detection

Enter a domain name to check:

www.google.com

Check

Prediction: normal