Spring 2023

Detecting Botnets Using Hidden Markov Model, Profile Hidden Markov Model and Network Flow Analysis

Rucha Mannikar

Detecting Botnets Using Hidden Markov Model, Profile Hidden Markov Model and Network Flow Analysis

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Rucha Mannikar

May 2023

The Designated Project Committee Approves the Project Titled

Detecting Botnets Using Hidden Markov Model, Profile Hidden Markov Model and Network Flow Analysis

by

Rucha Mannikar

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Fabio Di Troia	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Navrati Saxena	Department of Computer Science

ABSTRACT

Detecting Botnets Using Hidden Markov Model, Profile Hidden Markov Model and Network Flow Analysis

by Rucha Mannikar

Botnet is a network of infected computer systems called bots managed remotely by an attacker using bot controllers. Using distributed systems, botnets can be used for large-scale cyber attacks to execute unauthorized actions on the targeted system like phishing, distributed denial of service (DDoS), data theft, and crashing of servers. Common internet protocols used by normal systems for regular communication like hypertext transfer (HTTP) and internet relay chat (IRC) are also used by botnets. Thus, distinguishing botnet activity from normal activity can be challenging. To address this issue, this project proposes an approach to detect botnets using peculiar traits in the communication between command and control servers and bots. Patterns can be observed in botnet behavior like orchestrated attacks, heartbeat signals, or periodic distribution of commands. Hidden Markov Models (HMM) and Profile Hidden Markov Model (PHMM) are probabilistic models that can be trained on network traffic data to identify activity patterns that suggest botnet activity. In this project, HMM and PHMM are used to detect and classify botnets using publicly available datasets for real network data consisting of botnet traffic mixed with normal and background traffic. A comparative analysis of performance of HMM and PHMM is conducted in this project and the results show that HMM and PHMM can be useful in detecting botnets. PHMM outperforms HMM in terms of accuracy of botnet detection.

ACKNOWLEDGMENTS

I want to express my gratitude to San Jose State University and all the individuals who contributed to the successful completion of this project. I want to thank my project advisor, Dr. Fabio Di Troia for his valuable guidance, suggestions, and support throughout the project. I would also like to thank my committee members Dr. Navrati Saxena and Dr. Robert Chun for their support in the project. I'm grateful to the faculty members and college staff who facilitated access to the required resources and equipment for the project. I sincerely thank everyone who contributed to this project's success.

TABLE OF CONTENTS

CHAPTER

CHAPTER 1

Introduction

Botnet is a computer network infected by malware. The computer systems in the infected network called bots are managed by a remote attacker known as the botmaster. The botmaster performs regulated and harmonized attacks on a distributed platform enabling malicious activities on a large scale. Thus, botnet attacks are a potential hazard to Internet security. Botnets are particularly used to carry out DDoS (distributed denial of service) or personal information theft. Any computer system connected to the internet is vulnerable and at risk of a botnet attack.[1, 2]

The bots in the botnet communicate using standard networking protocols which makes detecting botnets challenging. However, certain communication patterns can be observed and deployed for botnet detection. The command and control (C&C) channels are typically used to transmit commands between the bots and the botmaster. It is required for the bots to connect to the botmaster at regular intervals using the C&C servers. This periodicity can be leveraged in botnet detection. Also, several other features can be extracted using the network traffic using flow-level characteristics like patterns in the number of packets or network-level features like degrees of centrality.[3, 4]

The goal of this project is to propose a HMM (Hidden Markov Model) and PHMM (Profile Hidden Markov Model) to identify botnet communication by using network flow data. This project tackles the following problems:

1. Given a network flow data consisting of packet captures from activities initiated by botnets, normal traffic and background traffic, is accurate detection and classification attainable?

2. The botnet traffic data is significantly low as compared to the normal and

background data resulting in a data imbalance issue. In what way can this problem be tackled?

3. How to extract meaningful features from given network flow data to train HMM and PHMM?

The structure of the report is as follows: Chapter 2 includes background details on the topics covered in the implementation of this project. Chapter 3 captures the relevant work related to this project. Chapter 4 describes the dataset utilized in the project. Chapter 5 discusses the implementation of the project in depth including steps like data pre-processing, feature extraction, HMM and PHMM training and evaluation. Chapter 6 consists of comparing and analyzing the results from the experiment.

CHAPTER 2

Background

This chapter elaborates on the background of the topics used in the implementation of this project. It discusses botnets, autocorrelation analysis, degrees of centrality, Hidden Markov Models, and Profile Hidden Markov Models

2.1 Botnets

A botnet is defined as a network of infected computers that perform malicious activities such as information theft, performing DDoS, sending spam, and so on. Botnet architecture consists of Botmaster, Bot Clients, C&C communication protocols and C&C server.[5]

1. *Botmaster*: The Botmaster manages and regulates communication with the bot clients. It controls the botnet army.

2. *Bot Clients*: Bot clients also called bots are the compromised computers that get added to the botnet and perform malicious activities in synchronization when commanded by the botmaster.

3. *C&C communication protocols*: These are communication protocols that botnet uses for sending and receiving commands from botmaster to the bots. The communication protocols can be Internet Relay Chat (IRC), Peer-To-Peer(P2P), Hypertext Transfer Protocol (HTTP) or hybrid such as combination of HTPP and P2P protocols.

4. *C&C servers*: The Botmaster sends commands through C&C servers and the bot clients periodically connect to the C&C servers to receive the commands from the botmaster.

Figure 1 illustrates the architecture along with the working of its components

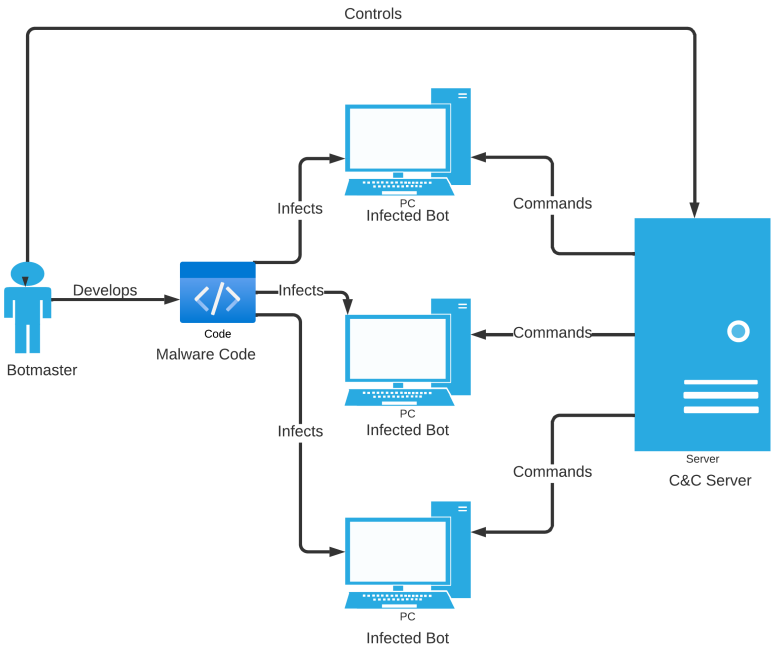

Figure 1: Botnet Architecture Overview.

The botnet life cycle consists of multiple phases. The first phase is the initial infection where the attacker finds vulnerabilities in the computer system and tries to attack it. It is followed by the secondary infection phase in which the system gets infected by downloading the malicious files containing dot binaries. The computer system gets compromised and turns into a bot thus joining the botnet. The next phases consist of these bots connecting to the C&C servers which are registered with Domain Name System (DNS) servers to avoid detection. The bots perform DNS lookup for connection. The connection phase after the DNS lookup phase is also called rallying. The new bots then wait and follow commands from the botmaster. This phase is called the malicious commands phase. Finally, the last phase is maintaining and updating the malware. When the malware is continuously updated executing newer methods, detecting the botnets becomes difficult. [5]

Figure 2 illustrates the botnet lifecycle phases in a sequential flow.

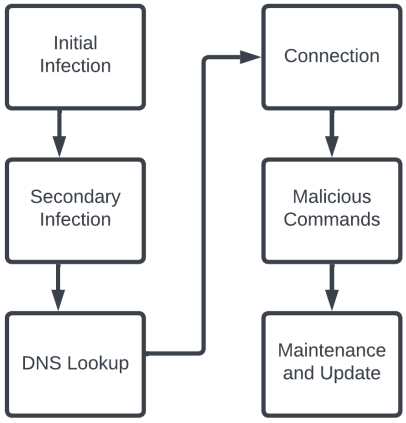

Figure 2: Botnet Lifecycle Phases.

2.2 Autocorrelation Analysis

Autocorrelation analysis is used to calculate and quantify the relationship of time series data points between distinct instances of time to detect a trend. It is used to thus find similarities with itself after specific time intervals. [3] The autocorrelation formula to calculate correlation between current data and data lagged by k units is as follows:

$$\rho(k) = \frac{\sum_{t=k+1}^{n}(x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}$$

where $\rho(k)$ is autocorrelation at lag k. x_t is data value at time t, $\bar{x}$ is the mean of the data, and n is the total number of data points. The numerator is the covariance between the original data and the k-unit lagged data. The denominator is the sum of the squared deviations of the original dataset. [6]

Figure 3 illustrates the Autocorrelation Function (ACF) plot for periodic and non-periodic data for 100 data points. The number of periods for the periodic data is 5. The x-axis is the lag and the y-axis is the autocorrelation function values. In periodic data, high ACF values are observed at lag positions. For non-periodic data, the values approach zero.

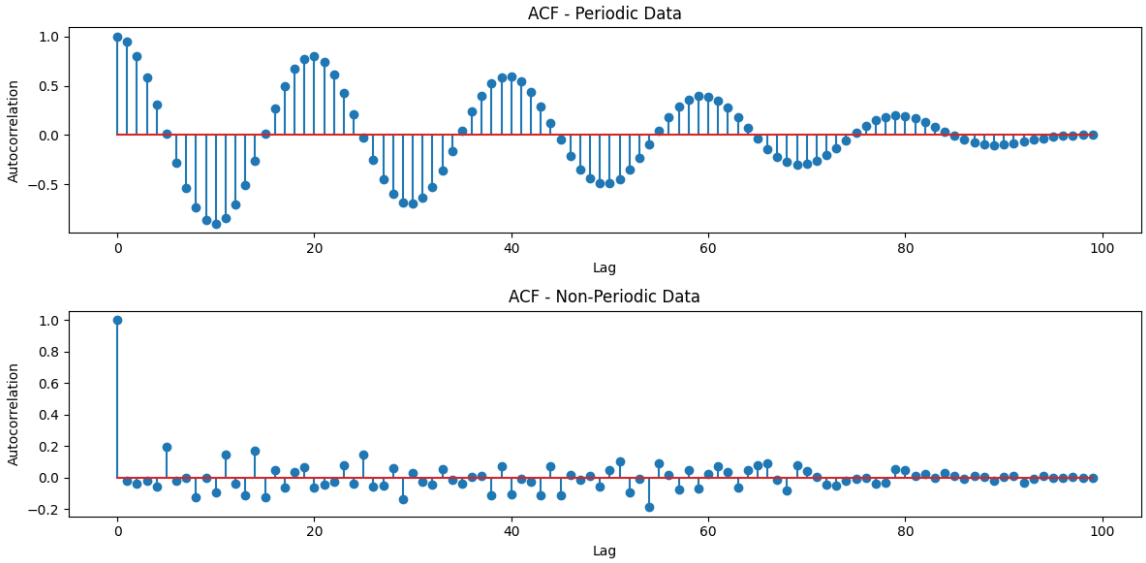

Figure 3: Plot of ACF for Periodic and Non-Periodic Data.

2.3 Degree Centrality

Degree centrality is used to quantify the connectivity of the node with other nodes. It measures the number of edges for a node. A well-connected node has a higher degree of centrality. The formula to calculate degree centrality is as follows:

$$C_D(v) = \frac{\text{number of edges incident to node } v}{\text{total number of nodes} - 1}$$

where $C_D(v)$ is the degree centrality of node v. Thus, in the context of network traffic, degree centrality can be used to calculate the importance of the computer system (node) in the network. [4]

2.4 Hidden Markov Models

The Hidden Markov Model (HMM) is a statistical model for Markovian systems where the future state depends only on the current state, irrespective of the historical states. In HMM, the states are hidden in contrast to the Markov Model. In HMM, the system consists of a Markov chain where the states are hidden and outputs are observable. HMM behaves like a state machine where every state has a probability

distribution associating the hidden state to the observation sequence. HMMs can be trained using the observation sequences as input and calculate the score for a test sequence. Thus, the probability of occurrence of such a sequence can be calculated. Table 1 consists of the standard notation for HMM. [7]

Table 1: Hidden Markov Model Notation [7]

Notation	Description
T	Length of the observation sequence
N	Number of hidden states in the model
M	Number of distinct observation symbols
Q	Distinct states of Markov process, denoted as $\{q_0, q_1, ..., q_{N-1}\}$
V	Possible observations, denoted as $\{0, 1, ..., M-1\}$
A	$N \times N$ State transition probability matrix
B	$N \times M$ Observation probability matrix
π	$1 \times N$ Initial state distribution matrix
O	Observation sequence, denoted as $(O_0, O_1, ..., O_{T-1})$

Figure 4 illustrates the diagram for HMM.

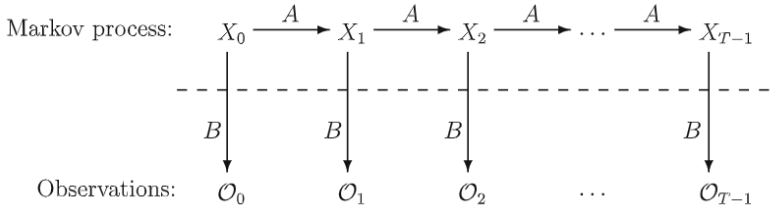

Figure 4: Hidden Markov Model [7].

HMMs are used to solve the following three standard problems:

Problem 1 Given a model $\lambda = (A, B, \pi)$ and an observation sequence O, we can determine the probability $P(O|\lambda)$. This is to calculate the score of the test observation sequence.

Problem 2 Given a model $\lambda = (A, B, \pi)$ and an observation sequence O, we

can determine an optimal state sequence for the Markov model by maximizing the expected number of correct states.

Problem 3 Given an observation sequence O and the parameter N, we can determine a model λ that maximizes the probability of O. Thus, we train a model to best fit an observation sequence. [7]

For this project, we use Problem 3 to train the HMM and Problem 1 to classify our test data.

2.5 Profile Hidden Markov Models

Profile Hidden Markov Models (PHMMs) are also statistical models and an extension of the HMMs where we can view PHMM as a series of HMMs. The training of PHMM several steps like creating multiple sequence alignment (MSA) using the training sequences. Every column in alignment is used to generate score corresponding to each position. PHMM has three states - match, insert and delete states. Insertion and deletion states are used for the gaps in aligned sequences. The assignment of transition probabilities is done to identify dependencies between adjoining positions.[8] Table 2 shows the standard notation used for PHMM.

Figure 5 illustrates the diagram for HMM.

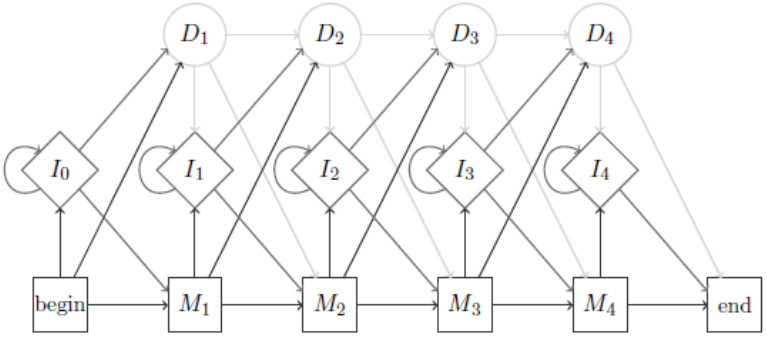

Figure 5: Profile Hidden Markov Model [8].

In this project we use the HMMER software package for training PHMM.

Table 2: Profile Hidden Markov Model Notation [8]

Notation	Description
X	Emitted symbols, $\{X_1, X_2, ..., X_m\}$ where $m \leq N + 1$
N	Number states in the model
M	Match states, $\{M_1, M_2, ..., M_N\}$
I	Insert states, $\{I_0, I_1, ..., I_N\}$
D	Delete states, $\{D_1, D_2, ..., D_N\}$
A	$N \times N$ State transition probability matrix
E	$N \times M$ Emission probability matrix
π	$1 \times N$ Initial state distribution
$a_{M_i M_{i+1}}$	Transition probability matrix from M_i to M_{i+1}
$\epsilon_{M_i}(k)$	Emission probability of symbol k at state M_i
λ	The PHMM, $\lambda = (A, E, \pi)$

CHAPTER 3

Relevant Work

There are different approaches for detecting botnets and can be categorized into four classes: signature-based, anomaly-based, Domain Name System (DNS) based, and data mining-based approaches. [9]

1. *Signature-based Detection*: In this method, traits in byte sequences of the packet captures are used for botnet identification by finding similarities with known signatures of the botnets. This technique performs well when the signature database is well-updated.[10]

2. *Anomaly-based Detection*: These techniques focus on the unusual traits demonstrated by the network traffic. They can be useful to identify new malicious activities which have not been encountered in the past. However, normal activities resulting in unusual spikes can be misclassified. [10]

3. *DNS-based Detection*: DNS server has the records of the IP addresses and the corresponding webistes. DSN-based techniques involve analyzing DNS traffic flow. Unusual spikes in the DNS queries can indicate botnet activity. [10]

4. *Data-mining-based Detection*: These approaches involve using data mining methods using the network traffic data by extracting meaningful features. These new features may involve statistical analysis of the network flow data or study of the network topology. In such methods, a labeled dataset can be used to train machine learning models to perform classification. [10]

Among all these techniques, the data mining techniques to detect botnets have several advantages. As botnets perform attacks on a large scale, data mining techniques can be useful to process the large volumes of data generated in the packet captures.

When newer types of malicious activities are encountered, they can be easily tracked with these methods as compared to signature-based techniques. As features are extracted subtle characteristics can be captured by data mining. In addition to this, these methods can help in predicting attacks based on past data. [1]

It can be observed that supervised learning techniques produce good results to detect botnets from network traffic data. [1] reports Naive Bayes, Random Forest, Artificial Neural Network, and Support Vector Machines to be effective models. [7] uses a novel approach of malware clustering based on HMM scores to detect malware. In this experiment, even previously unseen malware samples were detected with good accuracy. [3] uses the CTU-13 dataset for classifying botnets using deep learning models with good results. [11] uses an unsupervised machine learning approach to detect botnets using network traffic data. The algorithms used by [11] are clustering algorithms like k-means, x-means, and EM clustering. The results indicate that botnet flow data and normal data don't share similarities whereas botnet data shares some similarities. However, these techniques don't consider real-time botnet analysis. [12] uses a novel approach of detecting botnets in real-time using machine learning. Instead of calculating statistics from the network flow data, [12] uses only source and destination port number, number of packets, and total bytes transmitted. The machine learning algorithm chosen by [12] was a Decision tree classifier. However, this approach compromises model performance to give quick results. [13] uses a behavior-based approach to detect botnets by using the CTU-13 dataset. This study uses SMOTE (Synthetic Minority Oversampling Technique) for oversampling to balance data. [13] uses machine learning algorithms like Multilayer Perceptron (MLP), k-Nearest Neighbor (k-NN), and Support Vector Machine (SVM). Using the CTU-13 dataset, [14] uses a machine learning approach of detecting botnets using Decision Tree, Random Forest, and Adaboost. [14] uses a unique approach of a combination of

undersampling and oversampling to balance the data to get good results. [15] also uses the CTU-13 dataset along with machine learning algorithms like Gaussian naive Bayes (GNB), neural networks (NN), and decision trees (DT) with successful results. While these studies use the popular CTU-13 dataset, [16] focuses on detecting social bots which are automated accounts that may malicious links or spread fake news using deep learning techniques. Most of these studies use different machine learning models for comparative study. However, [17] uses an ensemble learning algorithm of majority voting from different supervised machine learning algorithms. Also, [17] too uses the CTU-13 dataset for the experiment to get good results. The studies discussed so far are not device-specific. However, [18] follows a novel approach to detecting botnets for internet-connected smartphones. Smartphone-specific features like permissions, API, and background services along with machine-learning techniques are used in this study. On similar lines, [19] focuses on Android botnets using machine learning algorithms like Naive Bayes and Random Forest. [20] uses machine learning models to detect botnets in the Internet of Things (IoT). This study also uses SMOTE to handle data balancing.

Thus, using machine learning models to detect and classify botnets using network flow data can achieve good results. Also, several researchers have used the CTU-13 dataset with machine learning approaches with successful outcomes. Some studies have also used SMOTE with CTU-13 dataset for oversampling to balance the data.

CHAPTER 4

Dataset

The dataset used for this project was the CTU-13 dataset which consists of botnet traffic. This dataset is publically available. The data was collected in 2011 by CTU University, Czech Republic. The dataset consists of three types of network traffic: botnet, normal, and background. In all there are thirteen files where each file has data for a specific scenario. For our project, we have used the bidirectional NetFlows data. The dataset has less proportion of botnet traffic data samples as compared to the normal and background traffic data.[21] Table 3 consists of the CTU-13 dataset scenarios with corresponding botnet types and the protocols used by each type.

Table 3: CTU 13 Dataset Scenarios with Botnet Types and Protocols [3]

Scenario No.	Botnet Type	Protocol Used
1, 2, 9	Neris	HTTP (Hypertext Transfer Protocol)
3, 4, 10, 11	Rbot	IRC (Internet Relay Chat)
5, 13	Virut	HTTP (Hypertext Transfer Protocol)
6	Donbot	IRC (Internet Relay Chat)
7	Sogou	HTTP (Hypertext Transfer Protocol)
8	Murlo	IRC (Internet Relay Chat)
12	NSIS.ay	P2P (Peer to Peer) Protocols

The CTU-13 dataset has 15 features for the bidirectional network flow data as shown in Table 4. The dataset consists of BINETFLOW-type file which was used as input for pre-processing.

Table 4: Description of Features of CTU-13 Dataset [3]

Feature	Description
StartTime	Timestamp of data indicating when it was recorded (hh:mm:ss)
Dur	Duration of associated network flow (in seconds)
Proto	Protocol used: tcp, udp, rdp, rtp, pim, icmp, ipx/spx, arp, igmp, rarp, unas, udt, esp, ipv6, ipv6-icmp
SrcAddr	Source IP address
Sport	Port number at the source
Dir	Direction of the network flow: -> (outgoing), <- (incoming), <-> (bidirectional), <?>(unknown), who (identity lookup), <? (unknown source), ?> (unknown destination)
DstAddr	Destination IP address
Dport	Destination Port number
State	Transaction state associated with the protocol
sTos	Source Type of Service
dTos	Destination Type of Service
TotPkts	Total transmitted packets
TotBytes	Total transmitted bytes
SrcBytes	Number of bytes transmitted from source to destination
Label	Traffic labels as normal, background, botnet

CHAPTER 5

Proposed Methodology and Implementation

The aim of this project is to classify network flow data as botnet activity or normal activity. Along with this, the goal is to classify the botnets if they belong to the seven botnet families (Neris, RBot, Virut, DonBot, Sogou, Murlo, and NSIS.ay), provided in the CTU-13 dataset. The proposed methodology is to extract features and train Hidden Markov Model (HMM) and Profile Hidden Markov Model (PHMM) to accurately classify data not known to these models. The entire process would have two phases - data pre-processing and model training with evaluation. The first phase will involve pre-processing the raw network flow data and extracting meaningful features from the 15 available features mentioned in Table 4. The second phase will have training, testing, and evaluating the HMM and PHMM models. Figure 6 illustrates the overview of the overall implementation process.

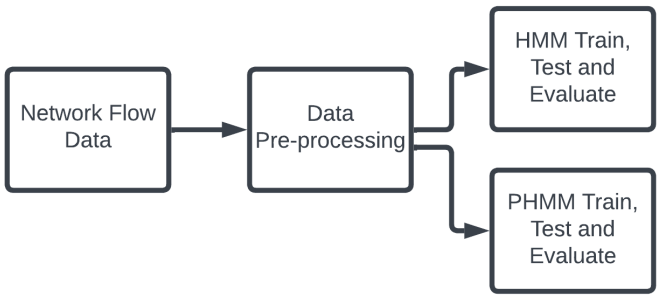

Figure 6: Implementation Overview.

5.1 Data Pre-processing

The data pre-processing phase includes the following steps: filtering network flow, creating a network graph, using the graph to create statistical data, calculating ACF from edges, finding the degree centrality of the source node, and finally normalizing the data. In addition to this, for PHMM we require another step of binning which converts the data into alphabets. These alphabets are further used to form sequences in multiple sequence alignment (MSA) format. These sequences are stored in FASTA format for input to PHMM. Figure 7 illustrates the overview of common data pre-processing required for HMM and PHMM.

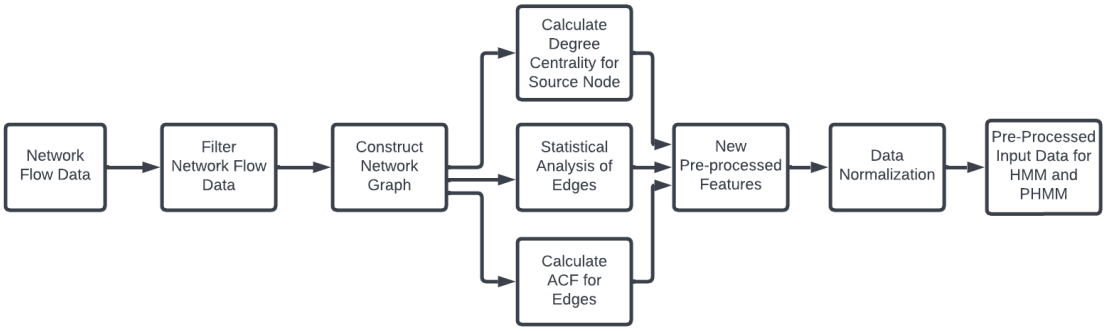

Figure 7: Data Pre-processing Overview.

5.1.1 Basic Data Cleaning

Basic data cleaning involves checking for null values, repetitions, and sorting the data in ascending order by StartTime.

5.1.2 Filter Network Flow

From the 15 available features mentioned in Table 4, we use only 10 significant features: StartTime, Dur, Proto, SrcAddr, DstAddr, State, TotPkts, TotBytes, SrcBytes, and Label. As per Table 3, the protocols used by botnets to communicate with the C&C server are TCP, UDP, HTTP, and ICMP. We keep connection states CON, URP, and FSPA_FPSA which are associated with these protocols [3]. Thus,

we remove unwanted protocols and states. Table 5 shows sample data entries after performing network flow filtering.

Table 5: Data after removing unwanted protocols and states

StartTime	Dur	Proto	SrcAddr	DstAddr	State	TotPkts
2011-08-15 16:43:20.935340	632.765381	udp	151.51.231.119	147.32.86.44	CON	32055
2011-08-15 16:43:20.937153	0.162553	tcp	147.32.84.229	66.56.30.27	CON	2
2011-08-15 16:43:20.949852	1793.589722	icmp	147.32.84.174	147.32.96.69	URP	96

TotBytes	SrcBytes	Label
8577197	8575244	flow=Background-UDP-Established
567	95	flow=Background-TCP-Established
56640	56640	flow=Background

5.1.3 Update Labels

The botnet labels in the CTU-13 dataset start with “From-Botnet”. On the contrary, the labels "To-Botnet" are associated with the flows from unknown computers to the botnet and are considered normal. All other labels are labeled as "Normal" [21]. Botnet labels containing “From-Botnet” are labeled as "Botnet" along with its types such as Neris, RBot, Virut, DonBot, Sogou, Murlo, and NSIS.ay. The old labels from the CTU-13 dataset are dropped.

5.1.4 Create Network Graph

A network graph is a representation of the computer network with nodes representing computer systems and the edges indicating the communication between these systems. For our project, the network graph is created with IPs from SrcAddr and DstAddr as the nodes and the edge representing the communication having features - protocol, duration, total bytes, total packets, state, timestamp, and label. Each edge

is a data table with multiple rows if communication between the two IP nodes is encountered multiple times with each row corresponding to one instance of communication. Figure 8 shows the network graph for few sample data points. Figure 10 shows

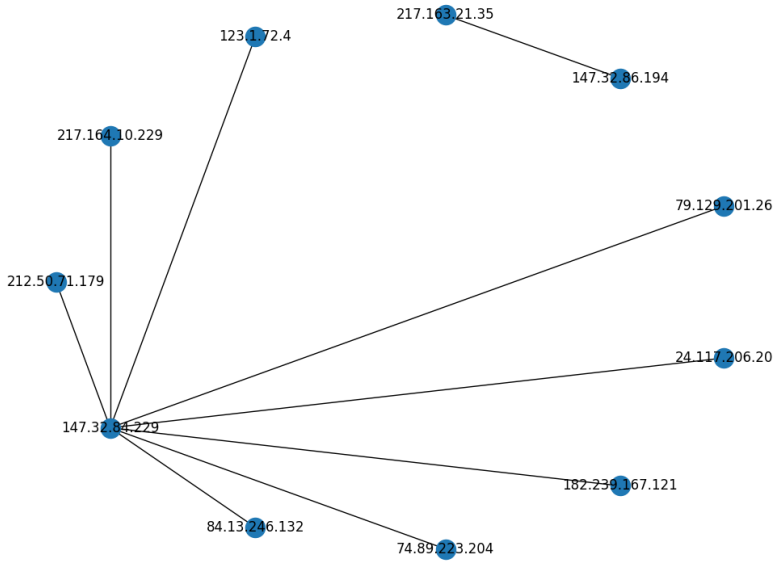

Figure 8: Sample Network Graph with few nodes.

the design for the network graph for three nodes. The edges store communication data with attributes such as StartTime, Dur, Proto, SrcAddr, DstAddr, State, TotPkts, TotBytes, SrcBytes, and Label. The steps involved in creating network graph are described in Figure 9.

5.1.5 Extracting Statistical Features from Network Graph

For finding meaningful features from our raw data attributes such as Dur, TotPkts, and TotBytes, statistical analysis is performed on these features. The statistics used for these features are mean, median, standard deviation, minimum, maximum, and range. Thus, each edge of the graph containing a table of network communication data is transformed into a single data row consisting of 18 new statistical features which we will consider for training our models. To get data from only important node

Algorithm 1: Pseudo-code for constructing network graph.

```
Input: Network flow data stored in an array (Array)
Output: Network Graph (G) consisting of nodes and edges
for each row in Array do
    Node1 = row.SourceIP
    Node2 = row.DestinationIP
    if Node1 not in Graph G then
        add Node1 to G
    if Node2 not in Graph G then
        add Node2 to G
    if edge does not exist between Node1 and Node2 then
        create a new edge between Node1 and Node2
    append row's flow attributes to the edge between Node1 and Node2
return G
```

Figure 9: Algorithm for constructing network graph.[3]

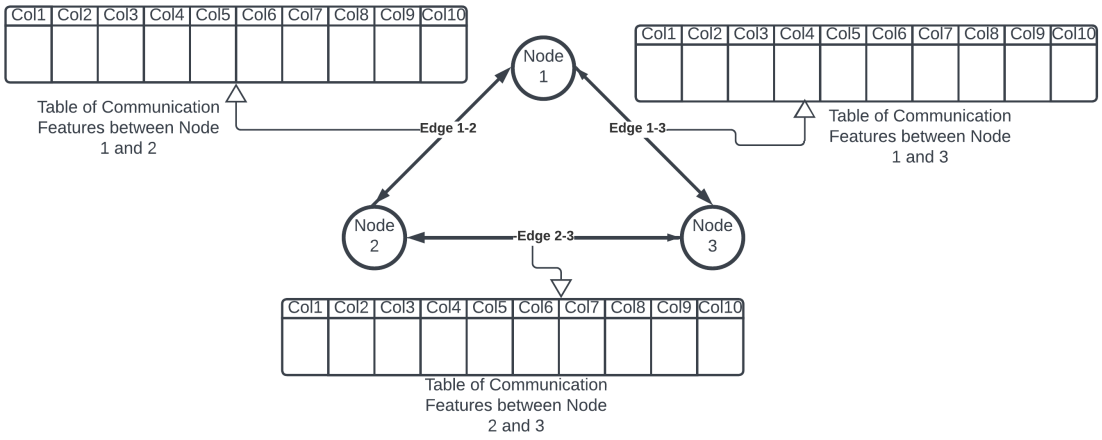

Figure 10: Network Graph design.

communications, we only consider edges that have more than 4 rows in the attribute table.

5.1.6 Extracting Autocorrelation Features from Network Graph

For each of the edges, we also calculate the ACF for attributes such as Dur, TotPkts, and TotBytes in the table of network communication. This helps us understand if the communication between the two nodes was periodic. We also calculate periodicity and strength for these features. Periodicity helps in detecting

repetitive traits whereas strength helps to quantify the intensity of the autocorrelation. Thus, we have 9 new autocorrelation features to be included for training the models. To get data from only important node communications, we only consider edges that have more than 4 rows in the attribute table. Figure 11 shows how periodicity is observed in the total bytes sent for an edge in our network graph in botnet communication. Figure 12 shows how periodicity is observed in the total duration for an edge in our network graph in botnet communication.

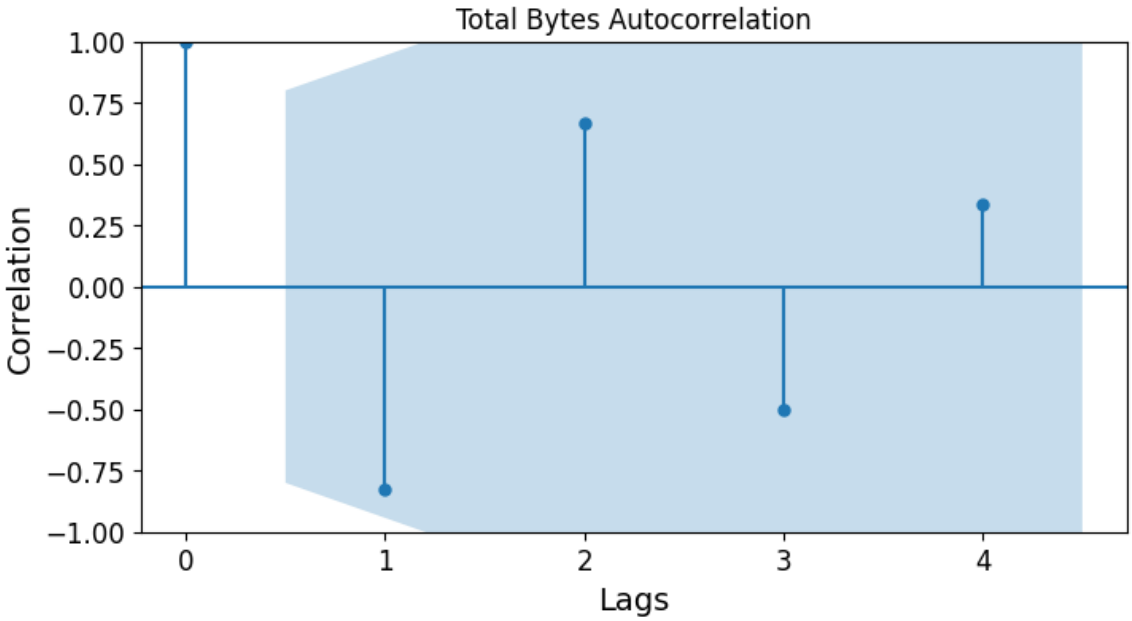

Figure 11: Total Bytes Autocorrelation Plot.

5.1.7 Extracting Topology Features from Network Graph

For each source node of the graph, we calculate the degree centrality of the node. It determines how important the particular node is in the network. It is used to quantify the connections a particular node has with other nodes. Botnets usually have a high degree of centrality. To get data from only important node communications, we only consider edges that have more than 4 rows in the attribute table. [4].

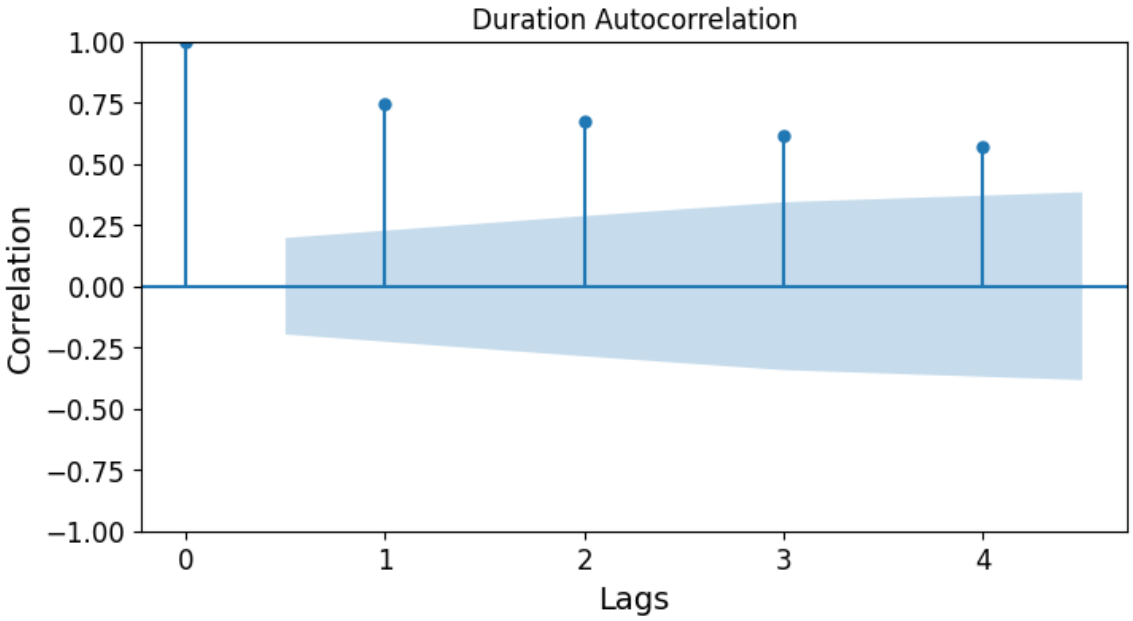

Figure 12: Total Duration Autocorrelation Plot.

5.1.8 Normalization

Data normalization is required for training the HMM using the hmmlearn library used for the experiment. This experiment uses the Gaussian HMM and the normalization process ascertains that data follows assumptions of the Gaussian distribution. Also, for PHMM we need to do additional pre-processing to convert data into the alphabet. Normalization helps in this step as well. In our project, MinMax normalization is used so that all values are in the range 0 to 1.

After performing data pre-processing, total of 30 features were extracted. These features are tabulated in Table 6

5.2 Hidden Markov Model (HMM)

To train the HMM, Gaussian HMM from the hmmlearn library was used in Python. For classifying data as botnet or normal, N (refer Table 1 for HMM Notation) was kept as 2. For classifying the botnets into different types N was kept as 8 as the CTU-13 dataset consists of 7 unique botnets and normal traffic. The data was split

Table 6: Extracted Features List after data pre-processing.

Sr. No.	Feature Name	Description	Category
1	DurMean	Duration Mean	Statistical Feature
2	DurMedian	Duration Median	Statistical Feature
3	DurStd	Duration Standard Deviation	Statistical Feature
4	DurMin	Duration Minimum	Statistical Feature
5	DurMax	Duration Maximum	Statistical Feature
6	DurRange	Duration Range	Statistical Feature
7	TotByteMean	Total Bytes Mean	Statistical Feature
8	TotByteMedian	Total Bytes Median	Statistical Feature
9	TotByteStd	Total Bytes Standard Deviation	Statistical Feature
10	TotByteMin	Total Bytes Minimum	Statistical Feature
11	TotByteMax	Total Bytes Maximum	Statistical Feature
12	TotByteRange	Total Bytes Range	Statistical Feature
13	TotPktMean	Total Packets Mean	Statistical Feature
14	TotPktMedian	Total Packets Median	Statistical Feature
15	TotPktStd	Total Packets Standard Deviation	Statistical Feature
16	TotPktMin	Total Packets Minimum	Statistical Feature
17	TotPktMax	Total Packets Maximum	Statistical Feature
18	TotPktRange	Total Packets Range	Statistical Feature
19	TotBytesACF	Total Bytes ACF	Autocorrelation Feature
20	TotBytesPeriodicity	Total Bytes Periodicity	Autocorrelation Feature
21	TotBytesStrength	Total Bytes Strength	Autocorrelation Feature
22	TotPktsACF	Total Packets ACF	Autocorrelation Feature
23	TotPktsPeriodicity	Total Packets Periodicity	Autocorrelation Feature
24	TotPktsStrength	Total Packets Strength	Autocorrelation Feature
25	DurACF	Duration ACF	Autocorrelation Feature
26	DurPeriodicity	Duration Periodicity	Autocorrelation Feature
27	DurStrength	Duration Strength	Autocorrelation Feature
28	Label	'Botnet' or 'Normal' Network Flow	Label
29	Class	Botnet type from 7 types dataset	Label
30	DegreeCentrality	Degree Centrality of Source Node	Topology Feature

into training and test data. The data was highly imbalanced with low botnet traffic flow data percentage as observed in practical real-life scenarios. To handle the data imbalance, a combination of oversampling and undersampling was used. The stratified sampling was used to ensure that all classes are represented in the same proportions

while training. The train-to-test ratio was kept at 80:20. The maximum number of iterations for HMM was kept at 1000. The test data consisted of 835 normal traffic samples and 837 botnet samples. Less data was used in HMM as compared to PHMM by combining undersampling and oversampling as HMM is a less complex model than PHMM and was trained effectively with less data achieving training accuracy of 95

5.3 Profile Hidden Markov Model (PHMM)

The data was split into train and test data with a ratio of 80:20. Stratified sampling was used to maintain an equal ratio of the botnet and normal traffic data while training the data. The data imbalance was handled by using SMOTE (Synthetic Minority Oversampling Technique). This oversampling technique creates synthetic data points with little variations in the original data. PHMM required additional data pre-processing. The original pre-processed data was scaled to the range 0 to 1. This data was converted to upper case alphabets (A-Z) using the process of binning where each alphabet corresponded to a continuous sub-range of 0 to 1. These features were concatenated together to form sequences. These sequences were converted to FASTA format files. The test data consisted of 2231 normal traffic sequences and 2852 botnet samples. Since PHMM is a complex model than HMM, more data was used by oversampling via SMOTE to achieve training accuracy of 97

The FASTA files were given as input to the PHMM model. For PHMM, the HMMER library was used. Seven separate PHMMs were trained - seven for separate botnet types. For testing, scores were calculated for each trained PHMM and the classification was performed based on the maximum score. Sequences which did not produce matches for any of the seven PHMM models were classified as normal traffic.

CHAPTER 6

Results

The proposed solution was implemented using HMM and PHMM. For HMM, the hmmlearn Python library was used whereas the HMMER tool was used for PHMM. The evaluation was performed using metrics like training, testing accuracies, precision, recall, and F1 Score. The results for these models were as follows as shown in Table 7, Table 8 and Table 9.

Table 7 shows the model evaluation for N=8. It can be observed that the highest accuracy is observed for Donbot followed by Neris. This could be due to the presence of more botnet data samples in the dataset. The lowest accuracy can be observed for detecting normal traffic. Figure 14 is the confusion matrix for multiclass classification for HMM. The test data consisted of 835 normal traffic samples and 837 botnet samples. It can be observed that though the botnets are being classified effectively, however normal traffic is not classified that well. This may be due to the fact that normal network traffic has more diversity and variability than botnet traffic. Diversity is introduced into normal traffic by factors like different protocols or user behavior. Botnet traffic, on the other hand, shows similar patterns. Table 8 shows model evaluation for N=2. Figure 16 shows the confusion matrix for N=2.

For PHMM, the normalized data were binned to associate with upper case English alphabets. In all, 25 bins were used to correspond with the first 25 alphabets. Bin edges were defined for each column in the dataset. Each of the train and test datasets were binned as shown in Figure 18. The data in the columns were then concatenated to form sequences as shown in Figure 19. These sequences were converted to FASTA format required for HMMER input. Each sequence was assigned a unique sequence number like 'Seq0' which was added as the header line for each sequence. The example of sequences can be seen Figure 20. These sequences were given as input to the

HMMER. True labels were stored in separate files for train and test sequences. The output from HMMER resulted in a text file as shown in Figure 21. This text file was parsed to store the data in Python dataframes. Out of all the values in the result file, the score represents the measure of the likelihood of the sequence belonging to the class. The scores from full sequences were used to determine the classes. The test sequences were given as input for all 7 PHMMs and the scores were compared to determine the probable class with maximum score values. The sequences which did not generate a match for any of the seven PHMMs trained for seven botnets were classified as normal. These predicted labels were compared against true labels to evaluate the model. Table 9 denotes the results obtained for PHMM. The highest accuracy was observed for NSIS.ay and the lowest for normal network traffic. For normal traffic, again this may be due to the fact that normal network traffic has more diversity and variability than botnet traffic. Figure 15 is the confusion matrix for multiclass classification for PHMM. The test data consisted of 2231 normal traffic sequences and 2852 botnet samples.

From Table 10, we can observe the average accuracies for train and test data for HMM and PHMM. We observe that PHMM performed better in terms of accuracy. Also, from Table 9, we can see that the accuracy for classifying normal traffic was better in the case of PHMM. This may be due to the fact that PHMM is a complex model as compared to HMM and could thus perform better. The comparison of the two models can be seen from Figure 17. It can be seen that accuracies are almost the same. However, comparing the precision for Table 9 and Table 7, it can be seen that the precision for different botnet types for PHMM is better than HMM. So we compare the average F1 scores for HMM and PHMM. F1 score uses both precision and recall for its calculation so it can be used for comparing the models. Table 11 summarizes the average F1 scores for HMM and PHMM. Figure 13 shows the comparison of F1

scores of both models. It can be observed that the F1 score for HMM is lower than PHMM. This is because the precision for HMM is lower than PHMM as normal traffic is being misclassified as botnet traffic owing to the fact that normal network traffic has more diversity and variability than botnet traffic.

Table 7: HMM Results for N=8.

Scenario No.	Botnet Type	Training Accuracy	Test Accuracy	Precision	Recall	F1-Score
1, 2, 9	Neris	0.9871	0.9844	0.7727	0.9883	0.8673
3, 4, 10, 11	Rbot	0.9811	0.9767	0.6727	0.9610	0.7914
5, 13	Virut	0.9402	0.9259	0.5823	0.9101	0.7102
6	Donbot	0.9989	0.9970	0.7058	0.9999	0.8275
7	Sogou	0.9853	0.9600	0.6646	0.9008	0.7649
8	Murlo	0.9623	0.9498	0.6784	0.9885	0.8046
12	NSIS.ay	0.9599	0.9480	0.7046	0.9801	0.8198
Normal	None (Normal Traffic)	0.8122	0.7600	0.9538	0.5449	0.6935

Table 8: HMM Results for N=2.

Traffic	Training Accuracy	Test Accuracy	Precision	Recall	F1-Score
Botnet	0.8319	0.8223	0.9177	0.7077	0.7991
Normal	0.8319	0.8223	0.7626	0.9366	0.8407

The PHMM model performed better than the HMM model considering the different evaluation metrics. This may be due to the fact that PHMM can register subtle details in a sequence alignment and like position-specific details and alignment characteristics. HMMs do not use positional details in observation sequences. However, execution of HMMs is faster than PHMMs.

Table 9: PHMM Results.

Scenario No.	Botnet Type	Training Accuracy	Test Accuracy	Precision	Recall	F1-Score
1, 2, 9	Neris	0.9932	0.9911	0.8455	0.9776	0.9068
3, 4, 10, 11	Rbot	0.9866	0.9715	0.6865	0.9355	0.7919
5, 13	Virut	0.9612	0.9580	0.7406	0.9916	0.8480
6	Donbot	0.9849	0.9739	0.8835	0.9125	0.8977
7	Sogou	0.9946	0.9964	0.9846	0.9901	0.9873
8	Murlo	0.9813	0.9933	0.8958	0.9247	0.9100
12	NSIS.ay	0.9970	0.9929	0.9074	0.9245	0.9158
Normal	None (Normal Traffic)	0.9047	0.8915	0.9535	0.7906	0.8644

Table 10: Comparison of Average Accuracies.

Model	Average Training Accuracy	Average Test Accuracy
HMM	0.9533	0.9377
PHMM	0.9754	0.9710

Table 11: Comparison of Average F1 Scores.

Model	Average F1 Score
HMM	0.7849
PHMM	0.8902

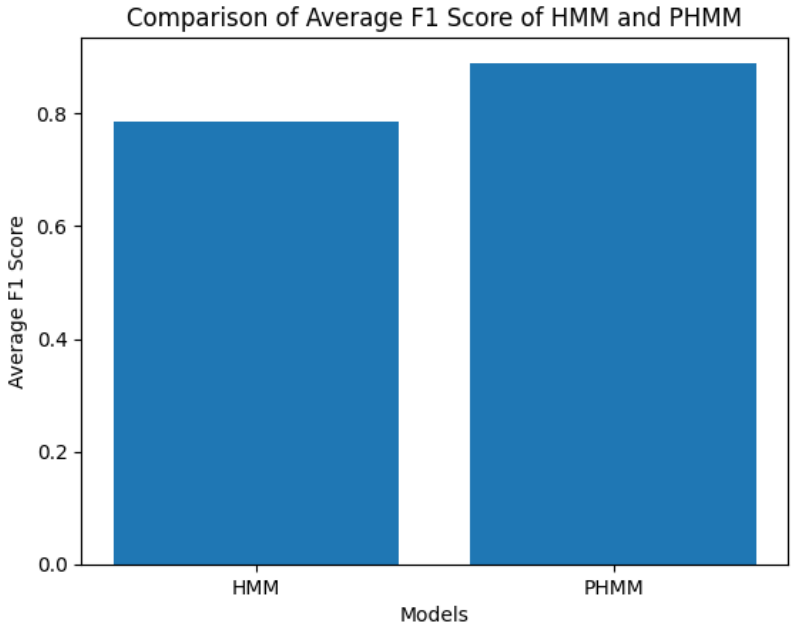

Figure 13: Average F1 Score Comparison for HMM and PHMM.

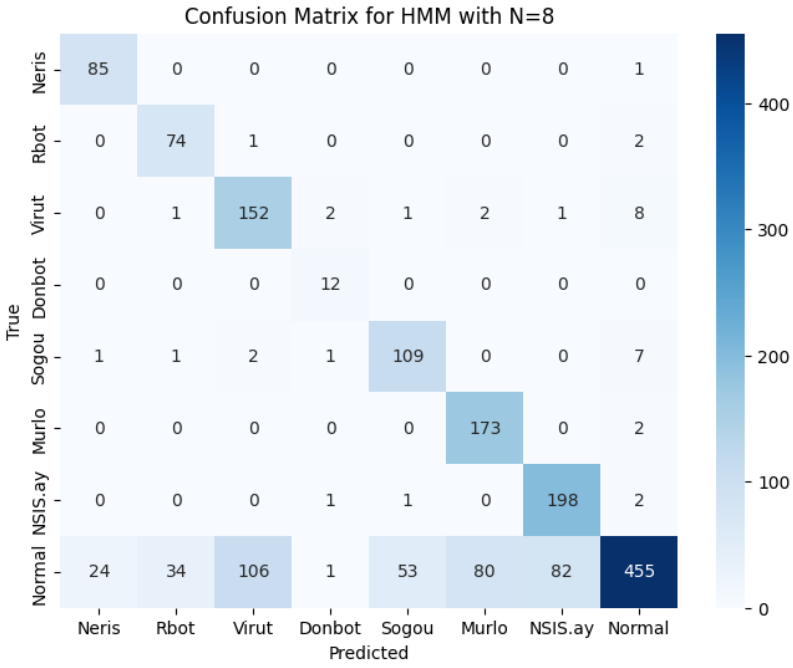

Figure 14: HMM Confusion Matrix for N=8.

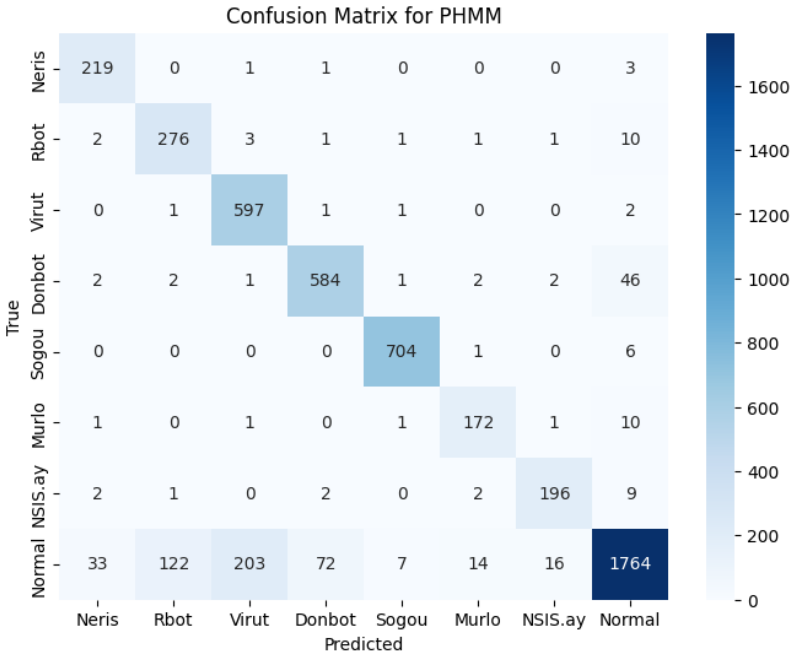

Figure 15: PHMM Confusion Matrix.

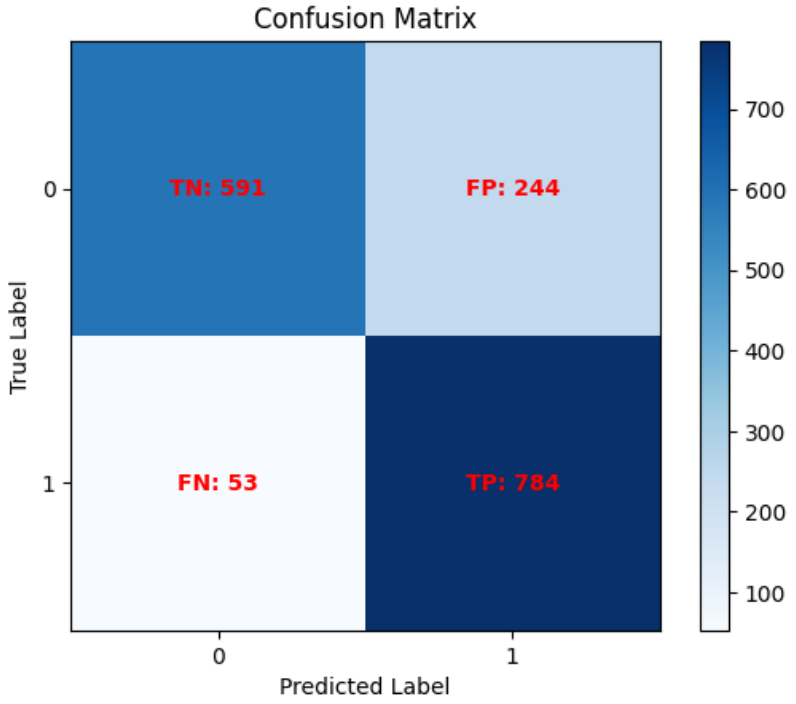

Figure 16: HMM Confusion Matrix for N=2.

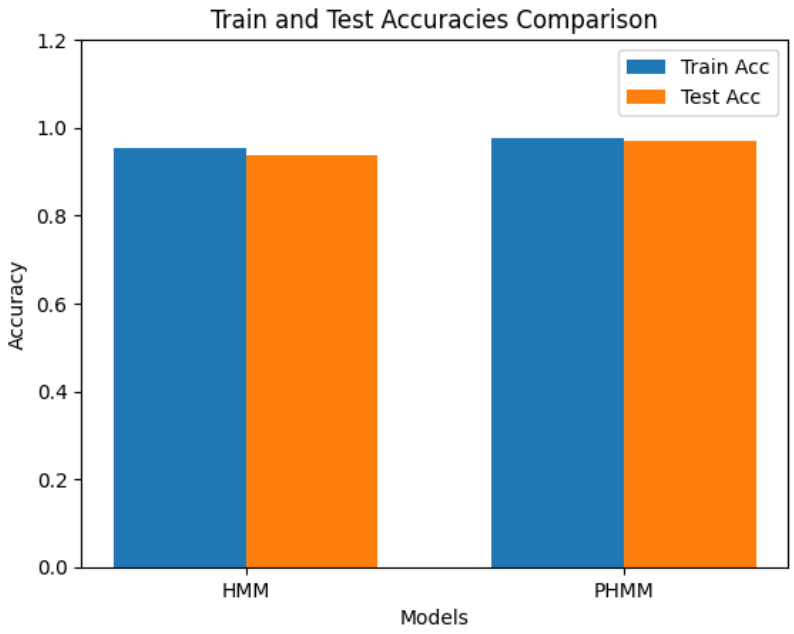

Figure 17: Accuracy comparsion for HMM and PHMM.

	DurMean	DurMedian	DurStd	DurMin	DurMax	DurRange	TotByteMean	TotByteMedian	\
0	N	R	Q	A	V	V	A	A	
1	G	A	R	A	Y	Y	A	A	
2	A	A	A	A	A	A	A	A	
3	A	A	A	A	A	A	A	A	
4	V	V	F	R	Y	H	A	A	
..	...	...	...	...	...	...	...	...	
95	A	A	A	A	A	A	A	A	
96	A	A	A	A	A	A	A	A	
97	A	A	A	A	A	A	A	A	
98	Q	S	R	A	X	X	A	A	
99	A	A	A	A	A	A	A	C	

	TotByteStd	TotByteMin	...	TotPktMean	TotPktMedian	TotPktStd	TotPktMin	\
0	A	A	...	A	A	A	A	
1	A	A	...	A	A	A	A	
2	A	A	...	A	A	A	A	
3	A	A	...	A	A	A	A	
4	A	A	...	A	A	A	A	
..	...	...	...	...	...	...	...	
95	A	A	...	A	A	A	A	
96	A	A	...	A	A	A	A	
97	A	A	...	A	A	A	A	
98	A	A	...	A	A	A	A	
99	A	J	...	A	C	A	J	

	TotPktMax	TotPktRange	TotBytesACF	TotPktsACF	DurACF	DegreeCentrality
0	A	A	H	K	T	Y
1	A	A	D	D	K	Y
2	A	A	H	H	J	B
3	A	A	I	J	H	A
4	A	A	K	E	D	Y
..	...	...	...	...	...	...
95	A	A	M	Q	G	Y
96	A	A	P	Q	R	Y
97	A	A	L	L	I	A
98	A	A	I	L	K	Y
99	A	A	K	K	J	A

Figure 18: Binning results for data columns.

0	NRQAVVAAAAAAAAAAAAHKTY
1	GARAYYAAAAAAAAAAAADDKY
2	AAAAAAAAAAAAAAAAAAHHJB
3	AAAAAAAAAAAAAAAAAAIJHA
4	VVFRYHAAAAAAAAAAAAKEDY
5	SVRAYYAAAAAAAAAAAAPNVY
6	KKRAXXAAAAAAAAAAAAHMGY
7	PSNEWSAAAAAAAAAAAAOQMY
8	AAAAAAAAAAAAAAAAAAJJHA
9	AAAABBAAAAAAAAAAAAIIHY

Figure 19: Sequences after concatenating binned columns.

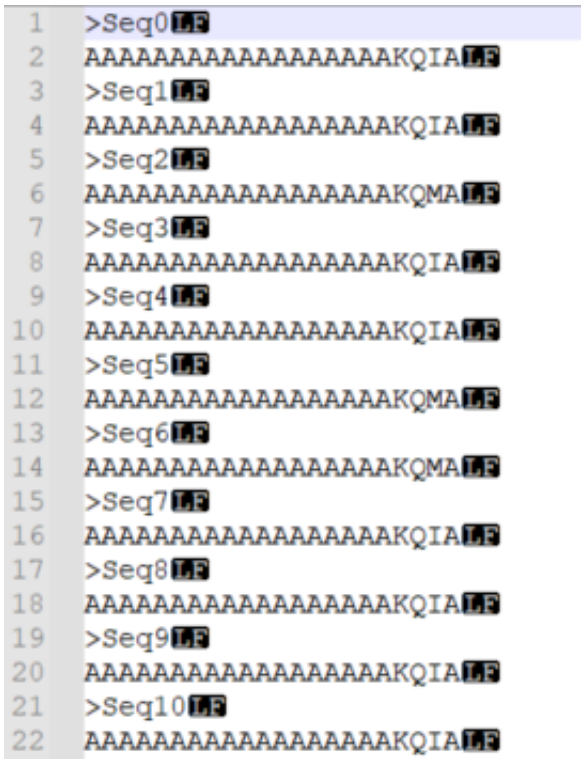

Figure 20: Example of FASTA Sequences.

```
# hmmsearch :: search profile(s) against a sequence database
# HMMER 3.3 (Nov 2019); http://hmmer.org/
# Copyright (C) 2019 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query HMM file:                  class1_hmm.hmm
# target sequence database:        test.fasta
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       train_class1  [M=22]
Scores for complete sequences (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence Description
    ------- ------ -----    ------- ------ -----   ---- --  -------- -----------
    4.4e-05   18.0  45.1    4.6e-05   17.9  45.1    1.1  1  Seq182
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq10
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq101
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq11
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq116
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq120
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq122
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq123
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq125
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq126
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq131
    0.00026   15.5  50.7    0.00029   15.4  50.7    1.1  1  Seq132
```

Figure 21: Output Text File for HMMER.

CHAPTER 7

Conclusion and Future Scope

In this project, the approach of using HMM and PHMM for botnet detection was implemented using network flow data. The project followed steps like data pre-processing, statistical, autocorrelation, and network topology feature extraction followed by model training, testing, and evaluation. It was seen that PHMM performed better as compared to HMM. PHMM performed well in classifying botnet families. HMM, on the other hand, gave an average performance. The models give good results, however, other models used in other studies like neural networks give better results than HMM and PHMM [3]. For future work, more topological features could be taken into consideration like betweenness centrality, clustering coefficient, and PageRank. Other models like SVM and Random Forest can be used to achieve good results in the future.

LIST OF REFERENCES

[1] M. Stevanovic and J. M. Pedersen, ''An efficient flow-based botnet detection using supervised machine learning,'' in *2014 International Conference on Computing, Networking and Communications (ICNC)*, Feb. 2014, pp. 797--801.

[2] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, ''Detecting P2P botnets through network behavior analysis and machine learning,'' in *2011 Ninth Annual International Conference on Privacy, Security and Trust*, July 2011, pp. 174--180.

[3] J. A. Lee and F. D. Troia, ''Detecting botnets through deep learning and network flow analysis,'' in *Artificial Intelligence for Cybersecurity*, M. Stamp, C. Aaron Visaggio, F. Mercaldo, and F. Di Troia, Eds. Cham: Springer International Publishing, 2022, pp. 85--105.

[4] S. Chowdhury, M. Khanzadeh, R. Akula, F. Zhang, S. Zhang, H. Medal, M. Marufuzzaman, and L. Bian, ''Botnet detection using graph-based feature clustering,'' *Journal of Big Data*, vol. 4, no. 1, pp. 1--23, May 2017.

[5] R. Limarunothai and M. A. Munlin, ''Trends and challenges of botnet architectures and detection techniques,'' *Journal of Information Science and Technology*, vol. 5, 2015.

[6] Will, ''Autocorrelation -- learn by marketing,'' https://www.learnbymarketing.com/definitions/autocorrelation/, accessed: 2023-5-15.

[7] C. Annachhatre, T. H. Austin, and M. Stamp, ''Hidden markov models for malware classification,'' *J. Comput. Virol. Hacking Tech.*, vol. 11, no. 2, pp. 59--73, May 2015.

[8] M. Stamp, *Introduction to Machine Learning with Applications in Information Security*. CRC Press, 2022. [Online]. Available: https://books.google.com/books?id=6pymEAAAQBAJ

[9] S. García, A. Zunino, and M. Campo, ''Survey on network-based botnet detection methods,'' *Secur. Commun. Netw.*, vol. 7, no. 5, pp. 878--903, May 2014.

[10] S. R. Mammunni and C. P. Sandhya, ''An overview of botnet and its detection techniques,'' *International Journal of Creative Research Thoughts*, vol. 8, Mar. 2020.

[11] W. Wu, J. Alvarez, C. Liu, and H.-M. Sun, ''Bot detection using unsupervised machine learning,'' *Microsystem Technologies*, vol. 24, pp. 209--217, 2018.

[12] J. Velasco-Mata, V. González-Castro, E. Fidalgo, and E. Alegre, ''Real-time botnet detection on large network bandwidths using machine learning,'' *Sci. Rep.*, vol. 13, no. 1, p. 4282, Mar. 2023.

[13] W. N. H. Ibrahim, S. Anuar, A. Selamat, O. Krejcar, R. Gonzalez Crespo, E. Herrera-Viedma, and H. Fujita, ''Multilayer framework for botnet detection using machine learning algorithms,'' *IEEE Access*, vol. 9, pp. 48 753--48 768, 2021.

[14] A. R. Vishwakarma, ''Network traffic based botnet detection using machine learning,'' Ph.D. dissertation, San Jose State University, 2020.

[15] S. Ryu, ''Comparison of machine learning algorithms and their ensembles for botnet detection,'' Open Access Theses, 2018. [Online]. Available: https://docs.lib.purdue.edu/open_access_theses/1451

[16] K. Hayawi, S. Saha, M. M. Masud, S. S. Mathew, and M. Kaosar, ''Social media bot detection with deep learning methods: a systematic review,'' *Neural Comput. Appl.*, vol. 35, no. 12, pp. 8903--8918, Apr. 2023.

[17] S. Baruah, D. J. Borah, and V. Deka, ''Detection of peer-to-peer botnet using machine learning techniques and ensemble learning algorithm,'' *International Journal of Information Security and Privacy (IJISP)*, vol. 17, no. 1, p. 16, 2023.

[18] S. Anwar, M. F. Zolkipli, V. Mezhuyev, and Z. Inayat, ''A smart framework for mobile botnet detection using static analysis,'' *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 14, no. 6, pp. 2591--2611, 2020.

[19] M. M. Rasheed, A. K. Faieq, and A. A. Hashim, ''Android botnet detection using machine learning,'' *Ingénierie des Systèmes d'Information*, vol. 25, no. 1, pp. 127--130, 2020.

[20] K. Alissa, T. Alyas, K. Zafar, Q. Abbas, N. Tabassum, and S. Sakib, ''Botnet attack detection in IoT using machine learning,'' *Comput. Intell. Neurosci.*, vol. 2022, p. 4515642, Oct. 2022.

[21] S. García, M. Grill, J. Stiborek, and A. Zunino, ''An empirical comparison of botnet detection methods,'' *Comput. Secur.*, vol. 45, pp. 100--123, Sept. 2014.