A STUDY OF DDOS ATTACK BY BOTNET INFECTED IOT DEVICES USING MULTIPLE MACHINE LEARNING CLASSIFIERS

A Major Project report submitted to

Cochin University of Science and Technology

In partial fulfilment for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

by

Kavya S (12170223)

Sarath T S (12170235)

Siddharth M (12170239)

Abhitha S (12170243)

Akhil Kuriakose (12170244)

Under the Supervision of

Mrs. Sreeja Nair MP

Department of Computer Science and Engineering

Cochin University of Science and Technology

November 2019

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that the project report entitled “**A Study of DDOS Attack by Botnet infected IoT devices using multiple Machine Learning Classifiers**” submitted by **Kavya S (12170223), Sarath T S (12170235), Siddharth M (12170239), Abhitha S (12170243), Akhil Kuriakose (12170244)** of Cochin University of Science and Technology towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science is a record of bona fide work carried out by them under my supervision and guidance during the academic year 2019-2020

Dr. Preetha Mathew	**Bindu P K**	**Mrs. Sreeja Nair**
Associate Professor	Head of Department	Assistant Professor
CUCEK	CUCEK	CUCEK

ACKNOWLEDGEMENT

We are grateful to our principal, Dr. Sunil Kumar N for the facilities provided by him for the preparation of this major project.

We are thankful to The Head of Department, Computer Science and Engineering, Bindu P K, for giving us all the support.

We are thankful to our project guide Mrs Sreeja Nair M P, for giving us all the support and valuable directions in overcoming the difficulties faced during the preparation of this major project.

We express our sincere thanks to our Project Coordinator, Dr. Preetha Mathew K, Associate Professor, Department of Computer Science and Engineering, for giving us innovative suggestions, timely advice, and suggestions during this endeavor.

Last, but not the least, we express our gratitude towards all staff members of Department of Computer Science and Engineering for their valuable advice and timely suggestions. We would also like to thank the faculties of Cochin University College of Engineering Kuttanad, CUSAT for their encouragement.

Kavya S (12170223)

Sarath T S (12170235)

Siddharth M (12170239)

Abhitha S (12170243)

Akhil Kuriakose (12170244)

ABSTRACT

An increasing number of Internet of Things (IoT) devices are connecting to the Internet, yet many of these devices are fundamentally insecure, exposing the Internet to a variety of attacks.

Botnets such as Mirai have used insecure consumer IoT devices to conduct distributed denial of service (DDoS) attacks on critical Internet infrastructure. This motivates the development of new techniques to automatically detect consumer IoT attack traffic. In this paper, we study DDos attack by botnet infected Iot devices using multiple machine learning classifiers.

DDoS detection in IoT network traffic with a variety of machine learning classifiers, including neural networks. These results indicate that home gateway routers or other network middleboxes could detect local IoT device sources of DDoS attacks using low-cost machine learning algorithms and traffic data that is flow-based and protocol-agnostic. We will be classifying all the data and provide graph for each and find the best algorithm that has the highest accuracy for detection of attack.

CONTENTS

1. INTRODUCTION

1.1 Internet of Things and Vulnerabilities:

The number of Internet of Things (IoT) devices is projected to grow from 8 billion in 2017 to 20 billion in 2020. Yet, many of these IoT devices are fundamentally insecure. One analysis of 10 currently popular IoT devices found 250 vulnerabilities, including open telnet ports, outdated Linux firmware, and unencrypted transmission of sensitive data.

The proliferation of insecure loT devices has resulted in a surge of loT botnet attacks on Internet infrastructure. In October 2016, the Mirai botnet commanded 100,000 loT devices (primarily CCTV cameras) to conduct a distributed denial of service (DDoS) attack against Dyn DNS infrastructure. Many popular websites, including Github, Amazon, Netflix, Twitter, CNN, and Paypal, were rendered inaccessible for several hours. In January 2017, the Mirai source code was publicly released; DDoS attacks using Mirai-derived IoT botnets have since increased in frequency and severity.

This growing threat motivates the development of new techniques to identify and block attack traffic from IoT botnets. Recent anomaly detection research has shown the promise of machine learning (ML) for identifying malicious Internet traffic. Yet, little effort has been made to engineer ML models with features specifically geared towards IoT device networks or IoT attack traffic. Fortunately, however, IoT traffic is often distinct from that of other Internet connected devices (e.g. laptops and smart phones). For example, IoT devices often communicate with a small finite set of endpoints rather than a large variety of web servers. IoT devices are also more likely to have repetitive network traffic patterns, such as regular network pings with small packets at fixed time intervals for logging purposes.

Building on this observation, we develop a machine learning pipeline that performs data collection, feature extraction, and binary classification for IoT traffic DDoS detection. The features are designed to capitalize on IoT-specific network behaviors, while also leveraging network flow characteristics such as packet length, inter-packet intervals, and protocol. We compare a variety of classifiers for attack study.

Given the lack of public datasets of consumer IoT attack traffic, we generate classifier training data by simulating a consumer IoT device network. We set up a local network comprised of a router, some popular consumer IoT devices for benign traffic, and some adversarial devices performing DoS attacks. Hopefully, we believe our classifiers would successfully identify attack traffic with an accuracy higher than 0.999. We plan to use datasets available in Kaggle or UCI Libraries.

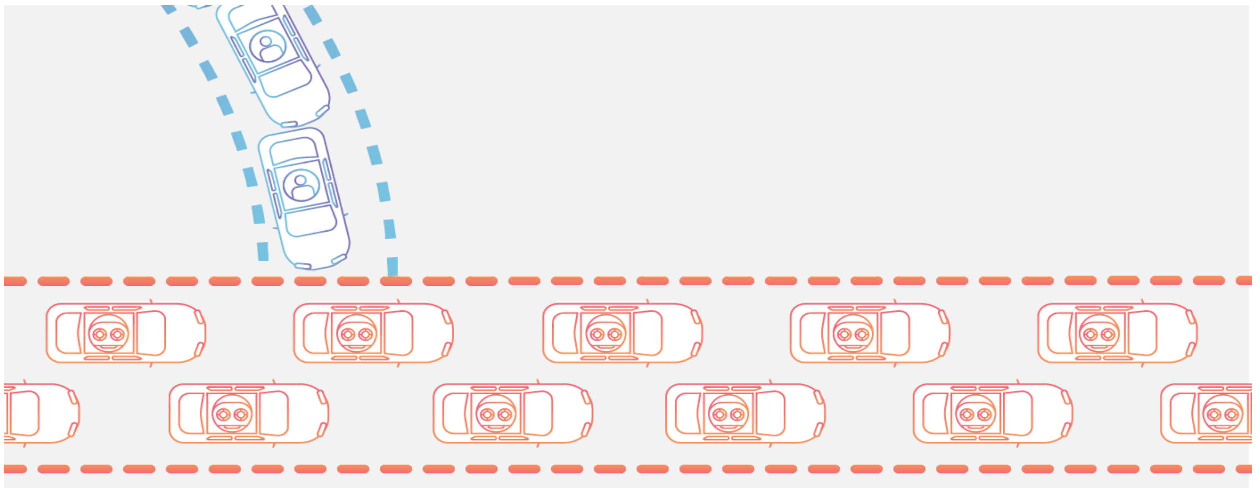
(REFERENCE: L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey", Computer Networks, vol. 54, no. 15, pp. 2787-2805, 2010.)

A little about the terms used:

A **distributed denial-of-service (DDoS)** attack is a malicious attempt to disrupt normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as IoT devices.

From a high level, a DDoS attack is like a traffic jam clogging up with highway, preventing regular traffic from arriving at its desired destination.



1.2 The Anatomy of a DDoS Attack

A simple principle governs a “**denial-of-service**” attack: attackers attempt to deny service to legitimate users. Some typical examples might include attackers overwhelming a server or cluster with requests, disrupting everyone’s access to the site or focusing the attack on a particular target who will be denied access.

With DDoS, the attacker usually has one of three goals:

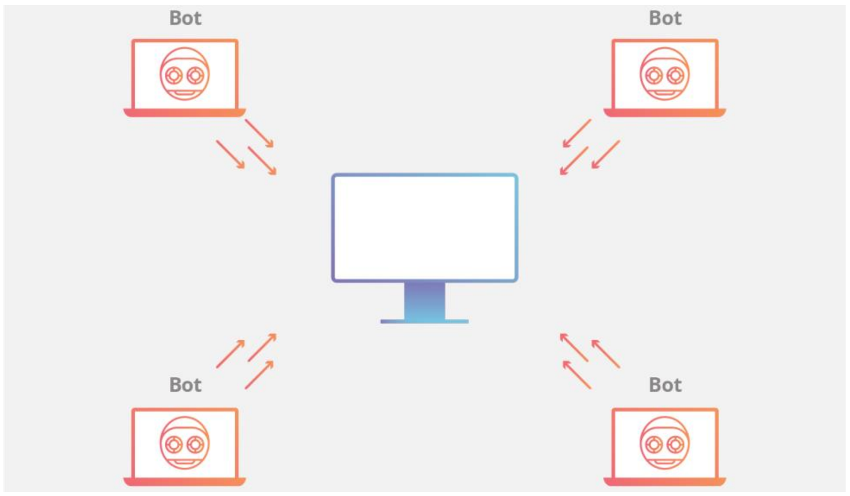
- To cause destruction or destructive change to network components
- To destroy configuration information
- To consume non-renewable or limited resources

DDoS attacks can be performed on their own or as part of a more massive attack on an organization. It usually targets bandwidth or processing resources like memory and CPU cycles. However, the type of DDoS attacks where we often see IoT devices used is a botnet attack.

1.3 What is a Botnet?

A botnet refers to a group of computers which have been infected by malware and have come under the control of a malicious actor. The term botnet is a portmanteau from the words robot and network and each infected device is called a bot. Botnets can be designed to accomplish illegal or malicious tasks including sending spam, stealing data, ransomware, fraudulently clicking on ads or distributed denial-of-service (DDoS) attacks.

While some malware, such as ransomware, will have a direct impact on the owner of the device, DDoS botnet malware can have different levels of visibility; some malware is designed to take total control of a device, while other malware runs silently as a background process while waiting silently for instructions from the attacker or "bot herder."



1.4 Why are botnets created?

Reasons for using a botnet ranges from activism to state-sponsored disruption, with many attacks being carried out simply for profit. Hiring botnet services online is relatively inexpensive, especially in relationship to the amount of damage they can cause. The barrier to creating a botnet is also low enough to make it a lucrative business for some software developers, especially in geographic locations where regulation and law enforcement are limited. This combination has led to a proliferation of online services offering attack-for-hire.

1.5 How is a botnet controlled?

A core characteristic of a botnet is the ability to receive updated instructions from the bot herder. The ability to communicate with each bot in the network allows the attacker to alternate attack vectors, change the targeted IP address, terminate an attack, and other customized actions. Botnet designs vary, but the control structures can be broken down into two general categories:

1.6 How do IoT devices become a botnet?

No one does their Internet banking through the wireless CCTV camera they put in the backyard to watch the bird feeder, but that doesn't mean the device is incapable of making the necessary network requests. The power of IoT devices coupled with weak or poorly configured security creates an opening for botnet malware to recruit new bots into the collective. An uptick in IoT devices has resulted in a new landscape for DDoS attacks, as many devices are poorly configured and vulnerable.

If an IoT device’s vulnerability is hardcoded into firmware, updates are more difficult. To mitigate risk, IoT devices with outdated firmware should be updated as default credentials commonly remain unchanged from the initial installation of the device. Many discount manufacturers of hardware are not incentivized to make their devices more secure, making the vulnerability posed from botnet malware to IoT devices remain an unsolved security risk.

1.7 How is an existing botnet disabled?

a. **Disable a botnet’s control centers:**

Botnets designed using a command-and-control schema can be more easily disabled once the control centers can be identified. Cutting off the head at the points of failure can take the whole botnet offline. As a result, system administrators and law enforcement officials focus on closing down the control centers of these botnets. This process is more difficult if the command center operates in a country where law enforcement is less capable or willing to intervene.

b. **Eliminate infection on individual devices:**

For individual computers, strategies to regain control over the machine include running antivirus software, reinstalling software from a safe backup, or starting over from a clean machine after reformatting the system. For IoT devices, strategies may include flashing the firmware, running a factory reset or otherwise formatting the device. If these options are infeasible, other strategies may be available from the device’s manufacturer or a system administrator.

2. FEASIBILITY STUDY

Before starting the project, feasibility study is carried out to measure the viable of the system. Feasibility study is necessary to determine if creating a new or improved system is friendly with the cost, benefits, operation, technology and time. Following feasibility study is given as below:

a. Technical feasibility

Technical feasibility is one of the first studies that must be conducted after the project has been identified. Technical feasibility study includes the hardware and software devices. The required technologies include Wireshark for packet sniff, Machine learning classifiers which can be implemented using python and sklearn libraries, and datasets are available online and we also using our iot devices for generation of data. Hardware requirements are all satisfied as we plan to train data using NVIDIA GTX GEFORCE 1050 Ti Graphics card on Intel i7 notebook and for packet generation we plan to use Raspberry pi 3 Model B or use open source datasets available from Kaggle/UCL Libraries. We plan to use Google Colabs if needed or Anaconda Navigator for training purposes.

Languages and Libraries used: Python, Scikit-learn, Matplotlib, Pandas

b. Operational feasibility

Operational Feasibility is a measure of how well a proposed system work out for the goal and takes advantage of the opportunities identified during scope definition. The following points were considered for the project's operational feasibility:

· The packet sniff accurately.

· The libraries are all available in sklearn and can be implemented using python.

The datasets are all public and no corporate issues generated.

Economic feasibility

The purpose of economic feasibility is to determine the positive economic benefits that include quantification and identification. The system is economically feasible due to the availability of all requirements such as collection of data and further can be implemented if possible, in different operating system to enhance security.

c. Economic feasibility

The purpose of economic feasibility is to determine the positive economic benefits that include quantification and identification. The system is economically feasible due to the availability of all requirements such as collection of data and further can be implemented if possible in different operating system to enhance security.

3. SECURING DEVICES FROM BOTNETS:

a. Create secure passwords:

For many vulnerable devices, reducing exposure to botnet vulnerability can be as simple as changing the administrative credentials to something other than the default username and password. Creating a secure password makes brute force cracking difficult, creating a very secure password makes brute force cracking virtually impossible. For example, a device infected with the Mirai malware will scan IP addresses looking for responding devices. Once a device responds to a ping request, the bot will attempt to login to that found device with a preset list of default credentials. If the default password has been changed and a secure password has been implemented, the bot will give up and move on, looking for more vulnerable devices.

b. Allow only trusted execution of third-party code:

If you adopt the mobile phone model of software execution, only whitelisted applications may run, granting more control to kill software deemed as malicious, botnets included. Only an exploitation of the supervisor software (i.e. kernel) may result in exploitation of the device. This hinges on having a secure kernel in the first place, which most IoT devices do not have, and is more applicable to machines that are running third party software.

c. Periodic system wipe/restores:

Restoring to a known good state after a set time will remove any gunk a system has collected, botnet software included. This strategy, when used as a preventative measure, ensures even silently running malware gets thrown out with trash.

d. Implement good ingress and egress filtering practices:

Other more advanced strategies include filtering practices at network routers and firewalls. A principle of secure network design is layering: you have the least restriction around publicly accessible resources, while continually beefing up security for things you deem sensitive. Additionally, anything that crosses these boundaries has to be scrutinized: network traffic, usb drives, etc. Quality filtering practices increase the likelihood that DDoS malware and their methods of propagation and communication will be caught before entering or leaving the network.

If you are currently under attack, there are steps you can take to get out from under the pressure. If you are on Cloudflare already, you can follow these steps to mitigate your attack. The DDoS protection that we implement at Cloudflare is multifaceted in order to mitigate the many possible attack vectors. Learn more about Cloudflare's DDoS Protection.

4. BACKGROUND AND RELATED WORK

In this section, we present a brief background on network anomaly detection and middlebox limitations.

a. Network Anomaly Detection

Anomaly detection aims to identify patterns in data that do not conform to expected behavior. In the context of our work, anomaly detection techniques may be used to discern attack traffic from regular traffic. Simple threshold-based techniques are prone to incorrectly classifying normal traffic as anomalous traffic and are unable to adapt to the evolving nature of attacks. More sophisticated anomaly detection algorithms, particularly those using machine learning, can help minimize false positives. Such approaches include deep neural networks, which promise to outperform traditional machine learning techniques for sufficiently large datasets.

Anomaly detection has long been used in network intrusion detection systems (NIDS) for detecting unwanted behavior in non-IoT networks. The NIDS literature can therefore inform the choice of anomaly detection methods for IoT networks. In particular, the literature suggests nearest neighbor classifiers, support vector machines, and rule-based schemes like decision trees and random forests as promising approaches.

Although there are parallels between NIDS and IoT botnet detection, there has been little work tailoring anomaly detection specifically for IoT networks. Our underlying hypothesis is that IoT traffic is different from other types of network behavior. For example, while laptops and smart phones access a large number of web endpoints due to web browsing activity, IoT devices tend to send automated pings to a finite number of endpoints. IoT devices also tend to have a fixed number of states, so their network activity is more predictable and structured. For instance, a smart light bulb could have three states: “On,” “Off”, and “Connecting to Wi-Fi.” each with distinctive network traffic patterns.

b. Network Middlebox Limitations

Network middleboxes have limited memory and processing power, imposing constraints on the algorithmic techniques used for anomaly detection. The literature contains suggestions for how to meet these constraints. For example, some security researchers investigated the use of software defined networks to monitor network traffic at flow-level granularity. Their work suggests that using flow-based features can be effective in detecting security threats without incurring the high cost of deep-packet inspection. An anomaly detection framework for a consumer smart home gateway router should therefore have the following characteristics:

Lightweight Features. Routers must handle high bandwidth traffic, so any features generated must be lightweight. In order for an algorithm to scale to high bandwidth application, a given algorithm must rely on network flow statistics (how packets are sent) as opposed to deep packet inspection (what is in a packet).

Protocol Agnostic Features. Routers must process packets from a variety of protocols (e.g. TCP, UDP, HTTP, etc.), so the algorithm must consider packet features shared by all protocols.

Low Memory Implementation. Routers are only able to maintain limited state due to memory constraints; caching adds latency and complexity. Thus, an optimal algorithm is either stateless or requires storing flow information over short time windows only.

(REFERENCE: Machine Learning DDoS Detection for Consumer Internet of Things Devices by Rohan Doshi, Noah Apthorpe, Nick Feamster ,Department of Computer Science Princeton University Princeton, New Jersey, USA)

5. THREAT MODEL

Our threat model makes various assumptions about consumer IoT networks. We assume the network includes an on-path device, such as a home gateway router or other middlebox, that can observe traffic between consumer IoT devices on the local network (e.g. a smart home LAN) and the rest of the Internet. The device at this observation point can inspect, store, manipulate, and block any network traffic that crosses its path. All traffic between WiFi devices on the LAN or from devices to the Internet traverses this middlebox.

Our goal is to detect and study DoS attack traffic originating from devices within the smart home LAN. The DoS victim may be another device on the LAN or elsewhere on the Internet. Any device connected to the middlebox can send both network and attack traffic within the same time period. Each device is also capable of conducting a variety of different DoS attacks in series, and successive attacks can vary in duration. This reflects how a remote botnet command and control (C&C) may change orders. We assume that the time range of DoS attacks are roughly 1.5 minutes, a common duration for DoS attacks attempting to avoid detection.

6. ANOMALY DETECTION PIPELINE

In this section, we present a proposed machine learning DDoS detection framework for IoT network traffic. Our anomaly detection pipeline has four steps:

1. **Traffic Capture:** The traffic capture process records the source IP address, source port, destination IP address, destination port, packet size, and timestamp of all IP packets sent from smart home devices.
2. **Grouping of Packets by Device and Time:** Packets from each IoT device are separated by source IP address. Packets from each device are further divided into nonoverlapping time windows by timestamps recorded at the middlebox.
3. **Feature Extraction:** Stateless and stateful features are generated for each packet based on domain knowledge of IoT device behavior. The stateless features are predominantly packet header fields, while the stateful features are aggregate flow information over very short time windows, requiring limited memory to support on-router deployment.
4. **Binary Classification:** K-nearest neighbors, random forests, decision trees, support vector machines, and deep neural networks can differentiate normal traffic from DoS attack traffic with high accuracy.

6.1 Traffic Collection

We would be setting up an experimental consumer IoT device network to collect realistic benign and malicious IoT device traffic, we could configure a Raspberry Pi v3 as a WiFi access point to act as a middlebox. We could then connect several other IoT devices to the Raspberry Pi's WiFi network

To collect normal (non-DoS) traffic, we could interact with all IoT devices for 10 minutes and recorded pcap files, logging all packets sent during that time period. We can perform many interactions that would occur during regular device use, including streaming video to the server in HD and RD modes, turning devices on/off and installing firmware updates, collecting sensor measurements and sending the measurements to a cloud server for storage. We could then filter out all non-IoT traffic from the pcap recordings.

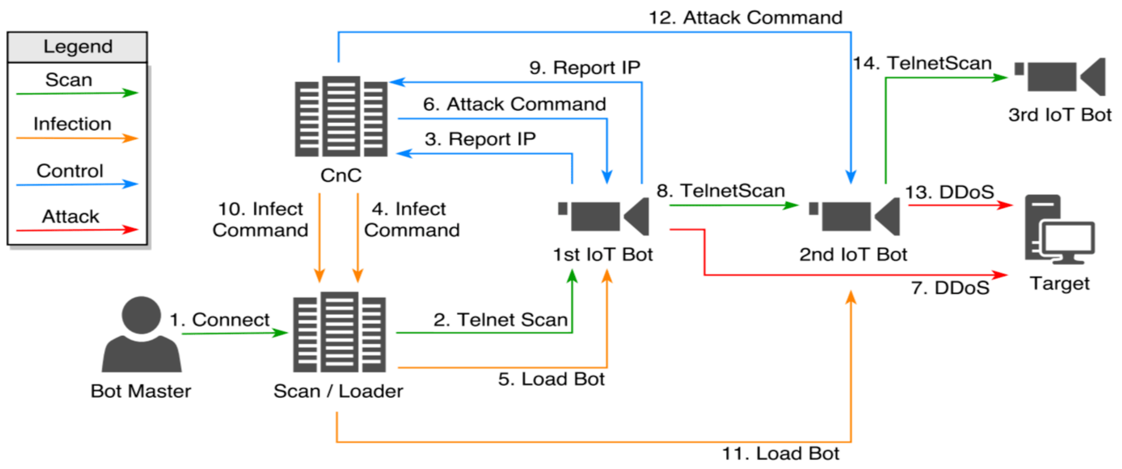
Collecting DoS traffic is more challenging. To avoid the security risks and complexity of running the real Mirai botnet code, we would be simulating the three most common classes of DoS attacks a Mirai-infected device will run: a TCP SYN flood, a UDP flood, and a HTTP GET flood. We could use a Kali Linux running on a laptop as the DoS source, and a Raspberry Pi running an Apache Web Server as the DoS victim. We could connect both devices via WiFi to our Raspberry Pi 3 access point. The DoS source then can target the victim's IP address with each class of DoS attack with a definite time separation.

6.2 Feature Engineering

We could explore two classes of features and analyze why they are relevant to differentiating normal and attack IoT traffic. Stateless features can be derived from flow-independent characteristics of individual packets. These features are generated without splitting the incoming traffic stream by IP source. Thus, these features are the most lightweight. Stateful features capture how network traffic evolves over time. There is inherent overhead in generating these features, as we split the network traffic into streams by device and divide the per-device streams into time windows. The time windows serve as a simple time-series representation of the devices' evolving network behavior. These features require aggregating statistics over multiple packets in a time window; the middlebox performing classification must retain state, but the amount of state can be limited by using short (e.g. 10-second) time windows.

7. IoT DDOS DETECTION PIPELINE:

7.1 How attack is generated:

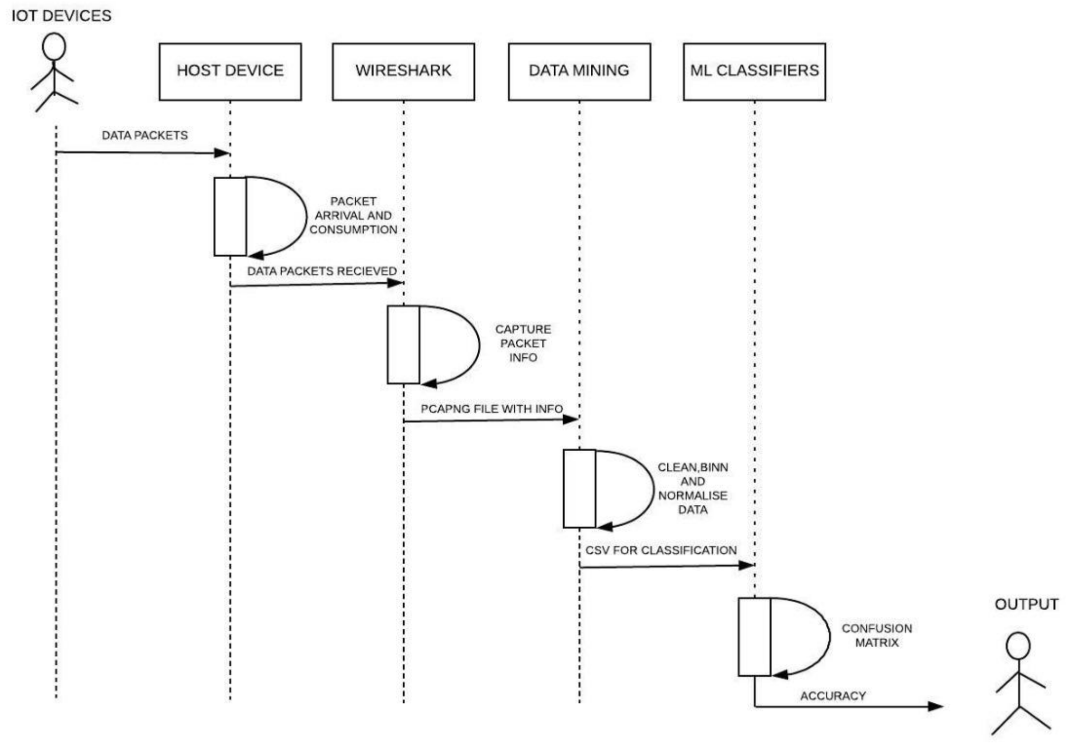


Infection and propagation occur by exploiting weak default security credentials found on many IoT devices running busybox, an embedded version of Linux. An attacker (botmaster) starts the process by connecting to the Scan/Loader server (step 1) and initiating. /loader to execute the scanner, c module, and scan the Internet for vulnerable IoT devices with Telnet services and ports 23 or 2323 open (step 2). Upon detecting a vulnerable device, the malware attempts to brute force a successful login using a list of 62 known default usernames and passwords. If successful, login credentials and device information are sent back to the C&C server, and will be used later by the Scan/Loader server to login and deliver the malware to the vulnerable device (step 3). An infect command is sent from the C&C server to the Scan/Loader server containing all necessary information such as login details, IP address, hardware architecture. Mirai supports multiple hardware architectures, including arm, mips, sparc and powerpc (step 4). The Scan/Loader server uses this information to login and instruct the vulnerable device to tftp or wget to the Scan/Loader server, download and execute the corresponding payload binary. Once executed, the first infected IoT device becomes part of the Mirai botnet and can communicate with the C&C server. The malware binary is removed and runs only in memory, to avoid detection (step 5). The botmaster can now issue attack commands, specifying parameters such as attack duration and target (step 6). The malware includes 10 DDoS attack types, including UDP flood (udp), Recursive DNS (dns), SYN packet flood (syn), ACK packet flood (ack), GRE flood (gre ip), which can be used to attack a target on the Internet (step 7). The first bot now attempts to repeat the infection process and propagate the botnet by scanning the Internet for additional vulnerable IoT devices with Telnet services and ports 23 or 2323 open (step 8). New vulnerable IoT device information is returned to the C&C server (step 9). A new infect command is issued to the Scan/Loader server (step 10). The appropriate hardware binary is loaded onto the newly discover vulnerable IoT device (step 11). The relevant attack command is issued from the C&C server (step 12). The attack is executed by the newly infected second bot, in conjunction with the first bot (step 13). Scanning for additional vulnerable IoT devices is repeated to further expand the botnet. (step 14).

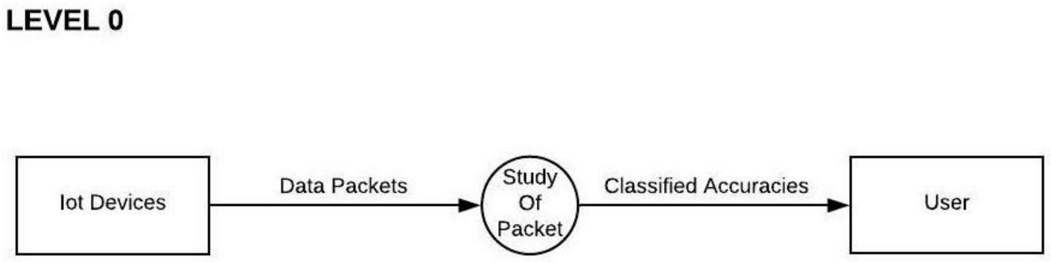
(REFERENCE: Botnet Detection in the Internet of Things using Deep Learning Approaches - Christopher D. McDermott; Farzan Majdani; Andrei V. Petrovski)

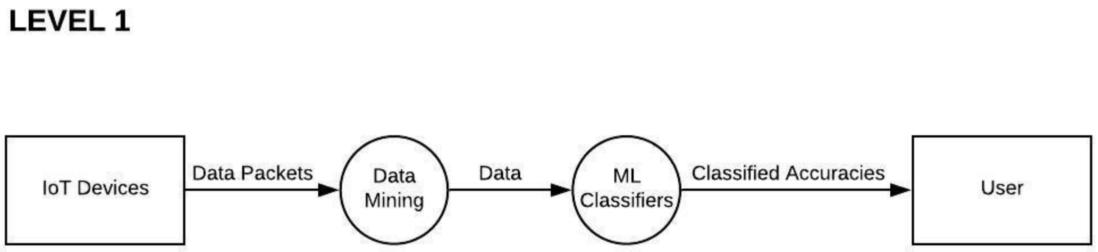
8. SYSTEM DESIGN

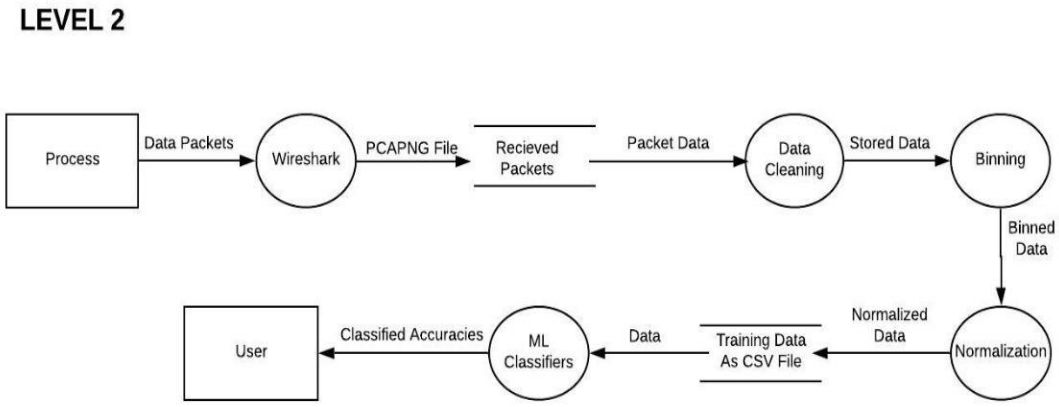
8.1 SEQUENCE DIAGRAM:



8.2 DFD:


LEVEL 0
Iot Devices
Data Packets
Study Of Packet
Classified Accuracies
User

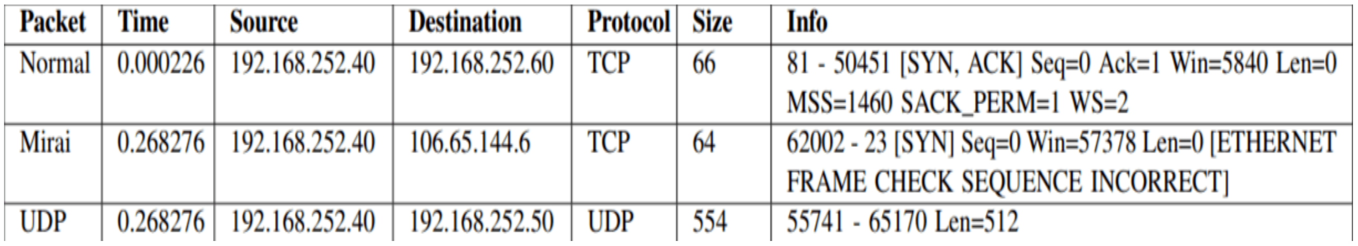

LEVEL 1
IoT Devices
Data Packets
Data Mining
Data
ML Classifiers
Classified Accuracies
User


LEVEL 2
Process
Data Packets
Wireshark
PCAPNG File
Recieved Packets
Packet Data
Data Cleaning
Stored Data
Binning
Binned Data
Normalization
Normalized Data
Training Data As CSV File
Data
ML Classifiers
Classified Accuracies
User

9. MACHINE LEARNING CLASSIFIERS:

Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Tree Classification, Random Forest Classifier, Naive Bayes Classifier, Artificial Neural Network (ANN)

FEATURE SET USED:



Packet	Time	Source	Destination	Protocol	Size	Info
Normal	0.000226	192.168.252.40	192.168.252.60	TCP	66	81 - 50451 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=2
Mirai	0.268276	192.168.252.40	106.65.144.6	TCP	64	62002 - 23 [SYN] Seq=0 Win=57378 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
UDP	0.268276	192.168.252.40	192.168.252.50	UDP	554	55741 - 65170 Len=512

The exact feature set which is used for machine learning training is:

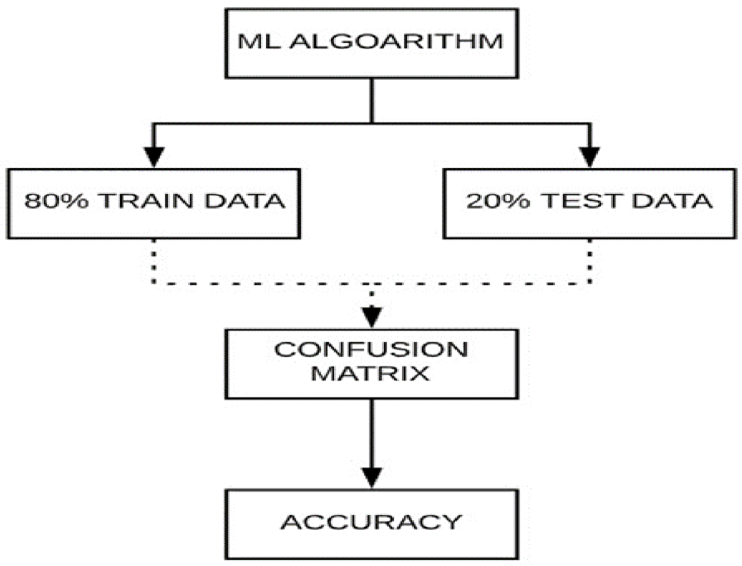
This is incoming packet structure we captured using Wireshark application. We use Packet, Time, Source, Destination, Protocol and Size for training. We may or may not choose Info fields based on improving accuracies of prediction.

The same feature set is being used for all the classifiers equal manner to test the one which gives the best accuracies.

We use help of Machine learning libraries sci-kit-learn and python language to get the accuracies from the data we collected earlier for training. So, we divide the data into 2 sets one is training and another testing set in ratio of 80:20 and check for accuracies using a confusion matrix. The confusion matrix first diagonal tells us about accurate result and other diagonal tells about inaccurate results from which we can derive the accuracies.

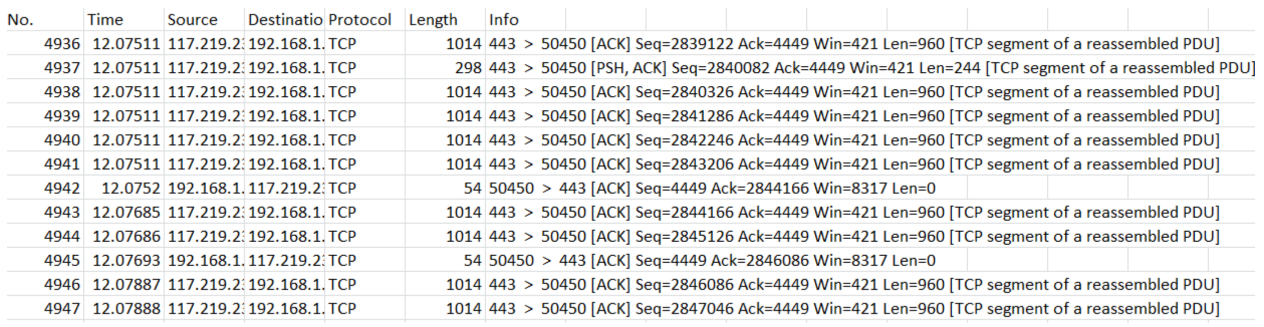
Before this step we have to get the data for training is CSV file which has all data in a scaled and normalized form. We need to perform the data mining step for this.

The data obtained in wireshark needs to be saved to a pcapng file and converted to csv file. This can’t be directly used for training as all data would be of different scales. So, we need to first clean the required data which can be done in a way we can provide labels and convert text content to relevant numbers. The data may be missing in few areas. So, we use concept of binning to input dummy data into appropriate spaces. Next the data would be of different scales and this needs to be converted to a single scale so that it can be provided into the activation function of ML algorithm. For all these we used pythons’ pandas library. Now after classifying we need to find the graph of accuracies which can be done using matplotlib. The below is how we train and test the data:



SAMPLE:

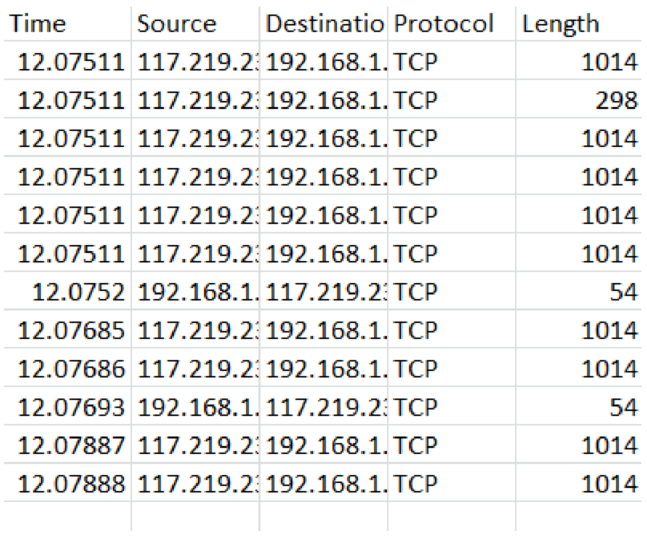
The below is the format in which we get our CSV file from PCAPNG format. This isn't clean data.



No.	Time	Source	Destinatio	Protocol	Length	Info
4936	12.07511	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2839122 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4937	12.07511	117.219.2	192.168.1.	TCP	298	443 > 50450 [PSH, ACK] Seq=2840082 Ack=4449 Win=421 Len=244 [TCP segment of a reassembled PDU]
4938	12.07511	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2840326 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4939	12.07511	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2841286 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4940	12.07511	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2842246 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4941	12.07511	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2843206 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4942	12.0752	192.168.1.	117.219.2	TCP	54	50450 > 443 [ACK] Seq=4449 Ack=2844166 Win=8317 Len=0
4943	12.07685	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2844166 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4944	12.07686	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2845126 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4945	12.07693	192.168.1.	117.219.2	TCP	54	50450 > 443 [ACK] Seq=4449 Ack=2846086 Win=8317 Len=0
4946	12.07887	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2846086 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]
4947	12.07888	117.219.2	192.168.1.	TCP	1014	443 > 50450 [ACK] Seq=2847046 Ack=4449 Win=421 Len=960 [TCP segment of a reassembled PDU]

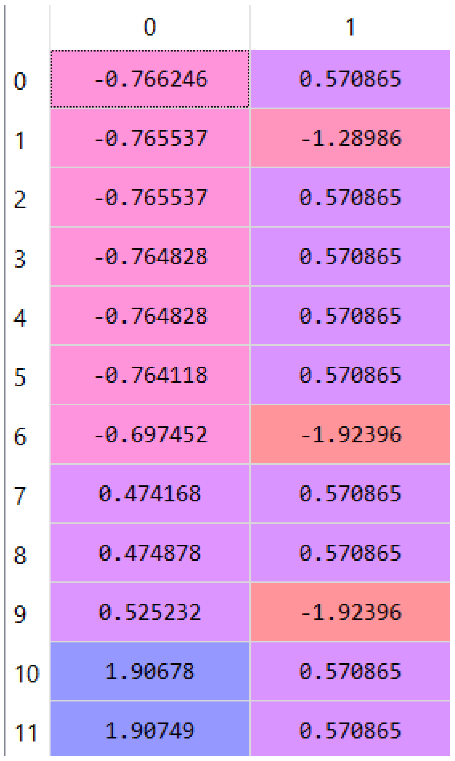
Now we have to clean the data using Pandas library and clean the data.

After cleaning: (We are using basic features here based upon our research we will increase feature set by also considering features from info)



Time	Source	Destinatio	Protocol	Length
12.07511	117.219.2	192.168.1.	TCP	1014
12.07511	117.219.2	192.168.1.	TCP	298
12.07511	117.219.2	192.168.1.	TCP	1014
12.07511	117.219.2	192.168.1.	TCP	1014
12.07511	117.219.2	192.168.1.	TCP	1014
12.07511	117.219.2	192.168.1.	TCP	1014
12.0752	192.168.1.	117.219.2	TCP	54
12.07685	117.219.2	192.168.1.	TCP	1014
12.07686	117.219.2	192.168.1.	TCP	1014
12.07693	192.168.1.	117.219.2	TCP	54
12.07887	117.219.2	192.168.1.	TCP	1014
12.07888	117.219.2	192.168.1.	TCP	1014

Now we need to bin the data and normalize it if needed. After feature scaling we have time and length as:



	0	1
0	-0.766246	0.570865
1	-0.765537	-1.28986
2	-0.765537	0.570865
3	-0.764828	0.570865
4	-0.764828	0.570865
5	-0.764118	0.570865
6	-0.697452	-1.92396
7	0.474168	0.570865
8	0.474878	0.570865
9	0.525232	-1.92396
10	1.90678	0.570865
11	1.90749	0.570865

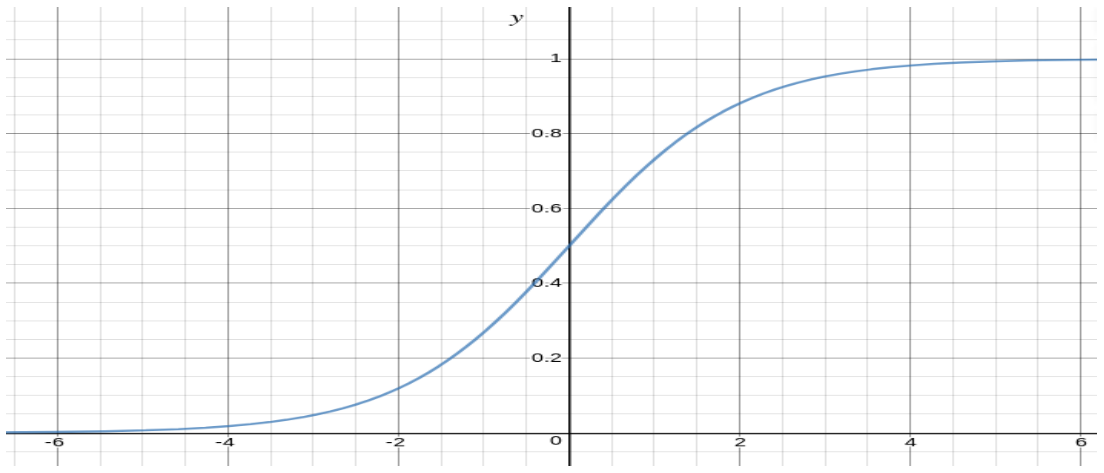
(SPYDER INTERFACE)

Now we use different classifiers to get the result of data.

Logistic Regression (Predictive Learning Model):

It is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

Activation function used is SIGMOID .



$$g(z) = \frac{1}{1 + e^{-z}}$$

Using python we can import the sklearn library as:

```
from sklearn.linear_model import LogisticRegression
```

Nearest Neighbor:

The k-nearest-neighbors algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbors), and has those neighbors vote, so whichever label the most of the neighbors have is the label for the new point (the "k" is the number of neighbors it checks). The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression). In the case of classification and regression, we saw that choosing the right K for our data is done by trying several Ks and picking the one that works.

Using python we can import the sklearn library as:

```
from sklearn.neighbors import KNeighborsClassifier
```

Support Vector Machines:

Support vector machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. Effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient. The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

Using python we can import the sklearn library as:

```
from sklearn import svm
```

Decision Trees:

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Using python we can import the sklearn library as:

```
from sklearn.tree import DecisionTreeClassifier
```

Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Using python we can import the sklearn library as:

```
from sklearn.ensemble import RandomForestClassifier
```

Naive Bayes Classifier (Generative Learning Model):

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Using python we can import the sklearn library as:

```
from sklearn.naive_bayes import GaussianNB
```

Neural Network:

A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand.

Using python we can import the sklearn library as:

```
from sklearn.neural_network import MLPClassifier
```

(**REFERENCE** : https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)

10. CONCLUSION

In this work, we would hopefully show that packet-level machine learning DoS detection can accurately distinguish normal and DoS attack traffic from consumer IoT devices.

We will be using a limited feature set to restrict computational overhead, important for real-time classification. Our choice of features was based on the hypothesis that network traffic patterns from consumer IoT devices differ from those of well-studied non-IoT networked devices.

We would test several ML classifiers on a dataset of normal and DoS attack traffic collected from an experimental consumer IoT device network. Among all classifiers, an algorithm or classifier with a test set accuracy higher would be selected for actual classification. At the end we would be providing graphs for all algorithm and show the algorithm that has highest accuacy for classification based on our research. These preliminary results motivate additional research into machine learning anomaly detection to protect networks from insecure IoT devices.

11. REFERENCES

a. Botnet Detection in the Internet of Things using Deep Learning Approaches - Christopher D. McDermott; Farzan Majdani; Andrei V. Petrovski - https://ieeexplore.ieee.org/document/8489489
b. L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey", Computer Networks, vol. 54, no. 15, pp. 2787-2805, 2010.
c. Machine Learning DDoS Detection for Consumer Internet of Things Devices by Rohan Doshi, Noah Apthorpe, Nick Feamster ,Department of Computer Science Princeton University Princeton, New Jersey, USA