Pre-Thesis -2 Report

IoT Botnet Detection using a Hybrid Quantum Machine Learning Model

Name and ID:

1)	Masroor Rahman -	19101213
2)	Md Muballigh Hossain Bhuyain -	19101289
3)	Reshad Karim Navid -	19101225
4)	Farnaz Fawad Hasan -	19101579
5)	Naima Ahmed Nup -	19101430

Supervisor: Dr. Mohammad Iqbal Hossain
Date of Submission: 18/09/2022

Department of Computer Science and Engineering
BRAC University

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Supervisor's Full Name & Signature:

Dr. Muhammad Iqbal Hossain

Abstract

Botnets are harmful software or malware that infect computers in various ways. Cyber thieves employ specific Trojan horses to breach the security of multiple users' computers and take control of a collection of infected workstations. These botnets are capable of committing a wide range of cybercrimes, including DDoS assaults, virus distribution, online fraud, and large-scale spam or phishing operations. Botnets have evolved from attacking PCs to vulnerable IoT devices. Botnets like Mirai have generated large DDoS attacks on well-known companies and devices and caused a lot of damage. So, it is very important to study these new IoT botnets to prevent future attacks.

Almost all ancillary devices in our homes and offices are internet enabled. This makes these devices susceptible to regular DDoS attacks through available open channels like NTP, LDAP and SNMP. In this paper we hope to look into how IoT based botnets work and try to explore quick methods for their detection. We hope to do so by using a hybrid quantum machine learning model which yields significantly better results compared to its classical counterpart.

Table of Contents

I. Introduction

1.1 Thoughts Behind two deadly IoT Botnets- Hajime vs. Mirai

Due to the rapid growth of technology during the 21st century, there are now more network-connected gadgets in people's residences, starting from clever home equipment to mild bulbs and thermostats that can be managed remotely.[20] These devices, however, are frequently targeted and concentrated on in order to establish massive, robust botnets. The Mirai [7] botnet, for example, registered with open Telnet services making use of default tool credentials. Large botnets and effective attacks have resulted from the combination of infection ease and rapid development of IoT devices. In 2016, many DDoS attacks were conducted using the Mirai Botnet [7], such as a 623 Gbps attack on krebson-security.com and a 1.2 Tbps attack on DNS company Dyn.

While plenty of studies have been achieved on the types of assaults executed by IoT botnets [16], in addition to the issues that permit them to flourish [6], [20], there may nonetheless be plenty to find about the underlying anatomy of susceptible IoT devices. Consider the following situations; What devices are at risk? Which architectures are most susceptible and where do these architectures tend to be more common? How big can an IoT botnet get globally right now? Is our privacy at risk? To resolve problems like these and lots of greater ones, we observe and examine differences between the ever-evolving Hajime and the Mirai botnet. The botnet Hajime is a contemporary of the Mirai botnet [6]. The characteristics of Hajime and Mirai is quite similar, however, they differ in three ways. First off, Hajime uses a well-known peer-to-peer (P2P) a distributed hash table (DHT) to distribute software updates to its bots rather than a centralized command-and-control (C&C) system. Secondly, unlike Mirai, Hajime supports a much broader variety of access methods, such as the latest Vault7 release [6]. Finally, Hajime provides documents via a proprietary protocol such as the transmission of a public key, that permits us to comply with the botnet's use of long-lived keys.

IoT botnets like these are simple yet lethal and elusive. Almost all significant recent DDoS attacks have incorporated botnets, which has caused service degradation, compromised data, lost money, and damaged brands for enterprises all over the world. We will be capable of using a machine learning approach containing a quantum layer to locate and save a likely IoT device from becoming infected by reviewing numerous datasets and studying similar IoT botnet's properties.

1.2 The Rise of the Mantis

Cloudflare revealed the greatest HTTPS DDoS assault it has ever withstood in June 2022, a 26 million request per second attack that is the largest on record. The Mantis Bots are so powerful that even with a small army of approximately 5000 bots it generated 26 million HTTPS requests per second [30]. Such attacks are very hard to conduct because of the need of large computational spaces and hefty expenditure in establishing secure TLS encrypted connections but this bot could prey on virtual machines and powerful servers unlike regular IoT botnets. Dynamic fingerprinting was used to mitigate the attack.

Newer IoT botnets like Mantis and Meris pose threat to cloud servers and it is a cause of concern. We believe that Quantum Machine Learning will work in defending against such botnets in a shorter time.

1.3 Research Problems

With the advancement of technology over the years security against botnets has increased but so have the intricacies of advanced botnet attacks. While countermeasures are adapting, the botmasters are also figuring out new ways to circumvent the new security measures. As a result, it's critical to develop algorithms that can learn about definite features of the botnet and detect them before they cause damage. Our model aims to tackle this issue by using a machine learning model that has a quantum layer embedded in it. This allows operations to be done much faster than a pure machine learning model as the quantum model can process some information simultaneously. While developing this model we expect to face some challenges and these are some of the hurdles we might face.

Finding a suitable dataset with proper documentation:
Although one can find numerous botnet datasets on the inter-web. It is quite, arduous to find a dataset that is well documented with their use cases outlined. Furthermore, it is an even greater task to find a dataset used in a particular research paper, where the dataset is pre-processed and models are run on it to prove the hypothesis of said research paper.

Research limited to Mirai focused algorithms:
The majority of the research that is now accessible is concentrated on the ISPs' available methodologies. The area of research also gets more constrained due to the limited availability of data sets and other methods of malware detection.It was found out that studying the algorithms of the Mirai domain did not result in many DNS based features [21]. However the use of DNS servers is seen predominantly in many IoT infected malicious malware to communicate with their CC servers or to employ fast-ux for address masking. This limitation in research limits us to find new opportunities to detect malicious botnets as such.

Quantum Computing Being a Relatively New Field:
Although quantum computing has become a buzz-word in the scientific community, the abundance of resources for specific usecases is quite sparce. So, researching and learning about such a new field is difficult when their is no clear "tutorial" to achieve exactly what we want.

Incorporating the Quantum Layer:
Although we are certain a quantum layer will certainly boost the performance of the existing classical model as seen in [8],[23] and [22], however, the process of meshing the quantum layer with the machine learning model to create a hybrid model which outperforms its predecessor is still unclear. Furthermore, what specific datasets and to add to

that what exactly we are looking for in said datasets also needs further study.

1.4 Research Objectives

This research aims to develop an advanced detection algorithm based on a hybrid quantum machine learning model. This model takes the classical machine learning model and adds a quantum layer to it, ensuring better performance overall. We hope our model aids in a more robust and more responsive detection system to protect IoT devices. Our objectives with this research include:

1. To deeply understand IoT and IoT devices.
2. To understand the inner workings and structure of the current IoT Botnets.
3. Develop a good understanding of quantum computing.
4. Understand machine learning.
5. Understand quantum machine learning.
6. To develop a model that combines both machine learning and quantum machine learning.
7. To try to acquire a better result.

II. Background

Although we have painted botnets as something that is inherently evil and malicious, [26] the internet as we now know it would not be conceivable without bots. Web crawlers, such as Googlebot, assist us in swiftly locating the most relevant content by browsing through millions of website of web pages in a flash. On all types of websites, chat rooms and dialog windows need chatbots for their operation. As proven by the case study, chatbots have progressed to the point where they can deceive actual humans as made apparent by the aptly named Cleverbot.[26]

In 2020, bot activity accounted for approximately 40% of internet traffic across the globe. Even though bots are critical in making the Internet a useful and powerful tool, they pose a massive threat to IoT devices, networks, ISPs, and users when created by criminals [26]. For instance, a botnet comprising mostly of embedded and IoT devices, called Mirai botnet, swept the Internet in late 2016 when it employed massive distributed denial-of-service (DDoS) operations to swamp several high-profile targets [7].

2.1 Related Works

2.1.1 Botnets Detection using Hybrid Quantum ML

In the papers below, hybrid quantum models are used in a variety of ways to detect or gain better insight into the working of various botnets.

i) Reduction of DDoS attacks using Mirai Botnets using Quantum IDS

According to [8], this paper aims to detect devices infected with botnets present in the IoT architecture in an attempt to thwart sophisticated attacks such as DDoS attacks carried out by Mirai Botnets. A Quantum Intervention Recognition Method is proposed where both classical and quantum techniques are used in tandem to detect infected IoT devices more efficiently and effectively by reducing the false positive rate along with conserving security and data integrity. The hybrid system proves to be a better alternative to its purely classical counterpart as the hybrid system outperforms it in both accuracy and speed of detection.

ii) Botnet Detection using DGA algorithm

In this paper [22], botnets were detected using a domain generation algorithm (DGA). Along with it, deep learning was implemented with the hybrid quantum model. Moreover, a differential privacy method was also utilized. The above model contains both classical deep learning layers in addition to quantum layers by integrating angle implanting and arbitrary layer circuits derived from the Pennylane framework. Although the classic DNS query analysis uses just the domain name analysis for examination and categorization, The domain names contained in DNS query data must to be regarded as private, as the domain names may reveal sensitive information. This model aims to produce a superior replacement for DNS using machine learning and AI. This has been done using 10 botnet DGA datasets and conducting experiments using both the host device's CPU and quantum devices. Although the computation time was quite long, the hybrid model yielded an accuracy of 92.4% whereas the classical model scored a mere accuracy rate of 88.4%.

iii) Detection Botnet DGA identification using Hybrid Deep Learning and Differential Privacy.

According to [23], The paper suggests a new way of detecting DGA-based botnets using hybrid quantum-classical deep learning models. Four features were picked from the Botnet DGA dataset in total, and the performance was compared between the hybrid model and the purely classical model. A quantum circuit layer was embedded into the deep learning model to make the hybrid model. The resulting model achieves results marginally superior in specific conditions to the classical non-hybrid model and depends on the noise values, random seed values, and the number of experiments performed.

iv) Cybersecurity botnet DGA detection with hybrid quantum-classical deep learning evaluation

According to [23], They contrasted hybrid quantum-classical deep learning models with traditional deep learning models for Botnet DGA classification. They utilized the Botnet DGA dataset for this. Their proposed hybrid quantum deep learning model significantly surpasses the classical model in a particular scenario; accuracy is up to 94.7% (n=100) and 93.9% (n=1,000). They found that the initial random seed values have an effect on the precision of our hybrid DL models. When using eight IBM quantum devices (ibmq 16 melbourne, ibmq 5 yorktown, ibmq athens, ibmq lima, ibmq quito, ibmq santiago, and ibmqx2) to implement noise models, the combination of angle embedding and strongly entangled ansatz produces high accuracy: the maximum and average accuracy are 94.7% and 91.4%, respectively.

From these papers, it is clear that a hybrid model, composed of both a quantum module and a classical machine learning model provides an overall better result when compared to just the classical model. Hence, threats can be detected quicker and thus prevention is made much more possible against such malicious attacks.

2.1.2 Botnets Detection using ML

i) Using Machine Learning Methodologies to Detect and Structurally Analyze Android Botnets.

According to [14], this paper shows the study of performance enhancement of a Machine Learning model is influenced by the choice of datasets and features, where the task of categorizing Linux Binaries as being potentionally harmful. The dataset utilizes 4 categories of IoT files which are system, application, botnet, and general malware files. These files are utilized for any ML model. They developed a system that was trained on these data and outperformed earlier approaches using a set of features that include static and dynamic network information. According to the article, training on system files or IoT application files is no longer adequate, but priming a model on IoT botnets can help identify zero-day assaults dramatically.

ii) Machine Learning used to Analyze Botnet DDoS Attacks.

According to [27], this research conducted an experimental investigation on ml methods for Botnet DDoS attack detection, with the algorithms examined including DecisionTree, SVM, NB, ANN and USML. The evaluation was performed using the KDD99 and UNBS-NB 15 datasets. It demonstrates that in terms of Accuracy, False Alarm Rate (FAR), Sensitivity, Specificity, FPR(False Positive Rate), AUC, and MCC, USML is very accurate at identifying botnet and regular network traffic.

iii) Determination of the effectiveness of varying ML Techniques on the Multivariate Categorization of Botnet Intrusion.

The goal of this paper[13] is to create a classifier that can detect anomalous traffic with comparatively higher general accuracy from the N_BaIoT dataset. To produce the outcome, four binary classifiers are evaluated and validated: Random Forests, Support Vector Machines, Extra Trees Classifiers, and Decision Trees. The results show that the classifiers perform very well when all of the classifiers are utilized to train and evaluate the irregularity within each device. To detect vulnerabilities on unrelated devices, Random Forests Classifier is very efficient.

iv) Analysis of the Micro Transport Protocol's Security with a Unreliable Receiver

According to [2], the three attacks listed in this paper's proposal for the uTP are the delay attack, lazy opt-ack attack, and opt-ack attack. They gave a comprehensive analysis and showed how serious these attacks were in terms of bandwidth consumption and packet volume. Additionally, they have demonstrated how the lazy opt-ack and opt-ack attacks may result in significant congestion as well as how the delay attack can rob extra bandwidth.

From these papers, we understand that the mentioned machine learning frameworks are effective in terms of accurately detecting bots but require categories of dataset files. The results are noticeably accurate compared to normal detection methods.

2.1.3 Botnet Attack Detection

i) The Defense System of a Botnet

The Botnet Defense System (BDS) is proposed in this research [28], a unique form of cyber security system that defends a network system from harmful botnets. A BDS distinguishes itself by combating harmful botnets with white-hat botnets. In September 2016, massive DDoS assaults brought down Twitter, Amazon, and other large websites. The assaults were carried out via Mirai botnets, a new form of botnet. Mirai's danger has been reduced in several ways. A BDS watches an IoT network, assesses the condition, and develops a strategy to battle hostile botnets. This paper provides a novel cyber security solution that leverages white-hat worms and controls white hat botnets to safeguard an IoT-system from harmful botnets in this research.

ii) A Hybrid Proposition for Botnet Attack Identification.

A hybrid strategy to detect botnet attacks is proposed in this paper [11]. It combines both administered and unadministered machine learning approaches in order to recognize attacks that are new in the network traffic. In order to achieve this, initially, they create a method in conjunction with the machine learning technique, then feed the outputs into an administered machine learning method, which categorizes the data as benign or hostile. The Intrusion Detection System (IDS) is important in detecting suspicious connections. Signature-based detection is one of the most often utilized methods. They wanted to create a hybrid solution in this paper by properly clustering normal and hostile data into several distinct groups, and then correctly classifying incoming traffic as benign or Mirai data. They used K-means clustering for machine learning which was unadministered and decision trees for administered machine learning. They present their preliminary work on combining unadministered and administered machine learning algorithms to solve the problem of IoT botnet attacks. They also put their method to the test on a data set, and the preliminary results show that it is effective.

iii) Analysis of Hajime: a friend or foe?

This paper illustrates the Hajime Botnet's assault strategy, infection procedure, and kind[3]. The Hajime is a P2P botnet that employs the.i and.atk extensions to carry out attacks. This botnet can be used for DDoS attacks, widely dispersed vulnerability scanning, extensive surveillance, and IoT bricking. The Hajime Botnet's netflow characteristics are discussed in the study. To identify the qualities, they used a C4.5 classifier, a greedy stepwise approach, the GA as an optimizer algorithm, and the machine learning algorithm C4.5.

2.1.4 Lightweight Botnet Detection using Fingerprinting.

In [9] it has been said that Flow-based botnet detection has been replaced in recent years and graph-based detection is also susceptible to exploitation. This has resulted in scalability concerns as well as increased time and space complexity. To avoid this communication graph and avoid scalability issues, a new method of detection of botnets has been proposed which considers the botnets' ability to utilize specific protocols and communicate with specific domains. Botnet Fingerprinting is a method or bot detection technique that uses attribute frequency distribution signatures to typify hosts' behavior patterns, learn friendly hosts' and bots' behaviors by applying supervised ML, and uses distances to labeled clusters or an ML system to classify incoming hosts as bots or friendly.

CTU-13 dataset was used to do the fingerprinting and it contains 13 instances of botnet infections. BotFP-Clus and BotFP-ML were the two approaches employed, the first one uses a clustering algorithm to cluster values and detect the outliers. The second one uses supervised learning. Supervised learning algorithms and neural networks take into consideration hyperparameters that must be changed to obtain the best classification results. Grid search generates and assesses a model for each hyperparameter combination specified in model tuning, then picks the most optimal hyperparamter to enhance a certain evaluation measure. It also employs (i) Logistic Regression (a statistical approach for calculating the probability of a variable that is dependent and has only two states) (ii) Support Vector Machines (SVM), which construct an optimal hyperplane for categorizing fresh samples using labeled training data, (iii) Random Forest Classifier and (iv) MLP classifier. In short, this method outperforms all the other ones with recalls ranging from 84% to 100% and precision ranging from 75% to 93%, depending on the approach.

2.1.5 Use of Deep Learning Methodologies to Detect IoT Botnets.

The proliferation of IoT-based DDoS attacks has been fueled by the expansion of IOT. McDermott's research [18] proposes a method for detecting botnet activity by utilizing Deep Learning to create a detection model based on a BLSTM-RNN (Bidirectional Long Short Term Memory-based Recurrent Neural Network). For text recognition, word Embedding is used and Attack Packets are translated to Tokenized Integer Format. The accuracy and loss of BLSTM-RNN are then compared to those of LSTM-RNN. Although the bidirectional strategy increases processing time and adds cost to each epoch, over time, it resulted to be a better model.

Following experimentation, the research effectively demonstrates the application of deep learning for botnet detection utilizing BLSTM-RNN in combination with word embedding. The accuracy of this approach was then compared to that of LSTM-RNN. For the four attack routes employed by the Mirai botnet, both models reported improved accuracy with a lower loss metrics. The constraints of previous specification or flow based detection approaches can be addressed since detection was done by supervising the packet flow and employing text recognition on elements that are ordinarily disregarded.

2.1.6 Feature Selection to Detect Botnets using Machine Learning Algorithms.

Alejandre [4] presents a method for doing feature selection to identify botnets during their C&C phase. One issue with feature selection is that academics have offered features based on their knowledge, but there is no way to assess these features because some of these characteristics may have a decreased detection amount than others. To deal with this, the article employs a feature set based on botnet connections throughout their C&C phase. To determine the collection of characteristics with the best detection rate, a Genetic Algorithm was used. Machine Learning Algorithms were also employed to help in the categorization of botnet connections. Additional experiments were carried out in order to obtain the optimal parameters in a GA. Experiments were carried out to find the optimal collection of attributes for each botnet under consideration.

As a consequence, it's shown that employing a GA as a classifier and optimizer algorithm to assess individuals in the GA to detect botnets in the C&C phase improves significantly over previous representative studies, with this research attaining the greatest detection accuracy using the same dataset. A feature drop was also seen when compared to the original feature vector. The detection rate of these attributes is higher than the original vector.

III. Dataset Description

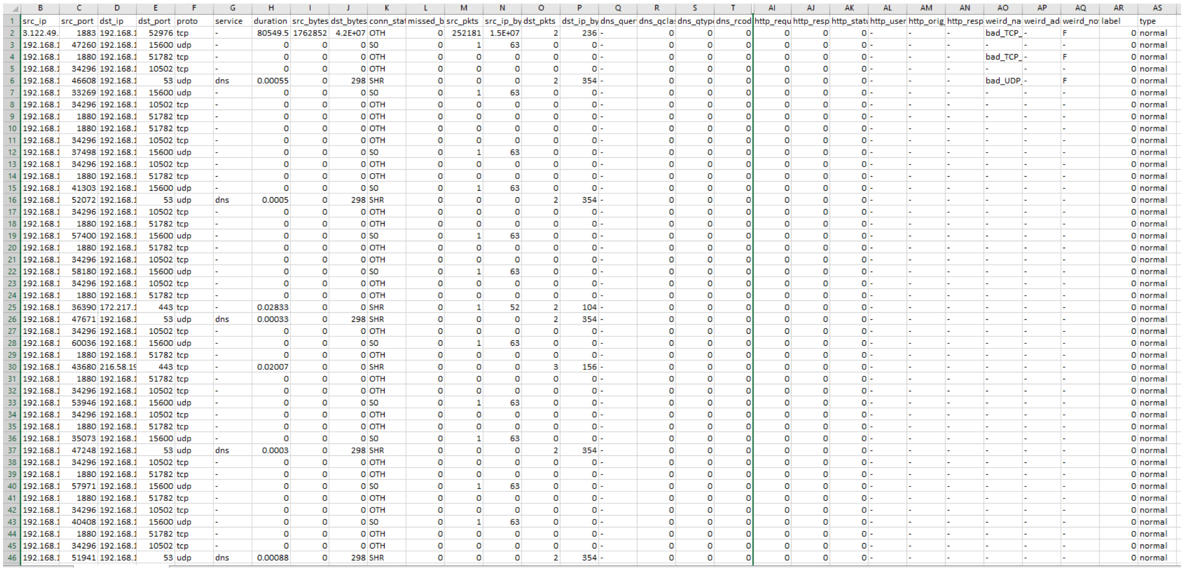
3.1 The ToN-IoT Dataset

In order to gauge the effectiveness, efficiency and fidelity of cybersecurity applications based on Artificial Intelligence oriented Deep Learning and ML algorithms, a new generation of datasets was developed by Dr Nour Moustafa, called the TON_IoT Datasets, UNSW Research, n.d. [25]. The datasets are referred to as ”ToN IoT” since they comprise a variety of data sources including Ubuntu 18 and 14 Transport Layer Security and Network Traffic Datasets, Windows 10 and 7 operating system datasets and telemetry datasets retrieved from IIoT sensors and IoT devices. The datasets were gathered from a large-scale, realistic network created at the Australian Defense Force Academy's IoT Labs Cyber Range, School of Engineering and Information Technology (SEIT), UNSW Canberra (ADFA). The IoT and IIot networks which constitute the industrial 4.0 network have a new testbed network. To supervise the connection amongst the three levels of Fog or Edge Systems and Cloud, IoT, the deployment of the testbed was done utilizing several VMs (virtual machines) and hosts of Kali, Linux and Windows Operating Systems. We took four instances from the aforementioned datasets, Network, Linux, Windows and one IoT device. After choosing the datasets we formulated a work plan to achieve the highest accuracy. We followed the same procedure for all the datasets. The step-by-step workflow is explained in-depth below:

- First, we imported the necessary libraries that are required for articulating the ML models. The libraries that we used are - NumPy, Pandas, Matplotlib, Seaborn and Sklearn. We also imported some models like Decision Tree Classifier, Logistic Regression, train_test_split and certain metrics like classification_report and accurary_report to report the precisions.
- After availing of our necessary libraries, we imported the dataset into a “df” or a data frame. All of our datasets were in CSV (comma-separated value) format.
- When the import was complete, we looked into the datasets to develop a rough insight into the features to grasp what we are dealing with.
- Empty Columns were firstly found and dropped.
- We encoded the features in order to have a unified dataset.
- Then we evaluated the features and developed a correlation matrix using *sns heatmap* first in order to understand which features have the most importance.
- After that we removed the features that had no relevance to make our dataset more efficient.
- Then we started splitting the data into independent X and dependent Y variables.
- We then started the Train Test Splitting where we trained 80% and tested 20%

- After our training was complete, we scaled our data in order to remove biases or deviance. This is known as Feature Scaling. The method we used was Min Max Scaling.
- Then we proceeded to transform the data and after that, we implemented the Machine Learning Models as discussed earlier.
- For every dataset, we ran Linear Regression, Decision Tree, Random Forest and Neural Network Classifier.
- Finally, we plotted the results in order to compare the precision of the Machine Learning models that we used.

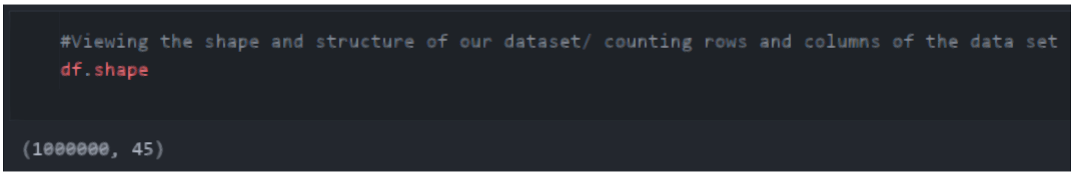
Network Intrusion Detection: Zeek (formerly Bro) is a Network Security Monitoring tool that was used to collect this dataset in the PCAP (packet capture) format. A snapshot of the dataset is attached below:



	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS
1	src_ip	src_port	dst_ip	dst_port	proto	service	duration	src_bytes	dst_bytes	conn_stat	missed_b	src_pkts	src_ip_by	dst_pkts	dst_ip_by	dns_quer	dns_qclas	dns_qtyp	dns_rcod	http_requ	http_resp	http_stat	http_user	http_orig	http_resp	weird_na	weird_ad	weird_no	label	type
2	3.122.49.	1883	192.168.1	52976	tcp	-	80549.5	1762852	4.2E+07	OTH	0	252181	1.5E+07	2	236	-	0	0	0	0	0	0	-	-	-	bad_TCP_	-	F	0	normal
3	192.168.1	47260	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
4	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	bad_TCP_	-	F	0	normal
5	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
6	192.168.1	46608	192.168.1	53	udp	dns	0.00055	0	298	SHR	0	0	0	2	354	-	0	0	0	0	0	0	-	-	-	bad_UDP_	-	F	0	normal
7	192.168.1	33269	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
8	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
9	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
10	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
11	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
12	192.168.1	37498	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
13	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
14	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
15	192.168.1	41303	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
16	192.168.1	52072	192.168.1	53	udp	dns	0.0005	0	298	SHR	0	0	0	2	354	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
17	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
18	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
19	192.168.1	57400	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
20	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
21	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
22	192.168.1	58180	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
23	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
24	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
25	192.168.1	36390	172.217.1	443	tcp	-	0.02833	0	0	SHR	0	1	52	2	104	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
26	192.168.1	47671	192.168.1	53	udp	dns	0.00033	0	298	SHR	0	0	0	2	354	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
27	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
28	192.168.1	60036	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
29	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
30	192.168.1	43680	216.58.19	443	tcp	-	0.02007	0	0	SHR	0	0	0	3	156	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
31	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
32	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
33	192.168.1	53946	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
34	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
35	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
36	192.168.1	35073	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
37	192.168.1	47248	192.168.1	53	udp	dns	0.0003	0	298	SHR	0	0	0	2	354	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
38	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
39	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
40	192.168.1	57971	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
41	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
42	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
43	192.168.1	40408	192.168.1	15600	udp	-	0	0	0	S0	0	1	63	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
44	192.168.1	1880	192.168.1	51782	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
45	192.168.1	34296	192.168.1	10502	tcp	-	0	0	0	OTH	0	0	0	0	0	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal
46	192.168.1	51941	192.168.1	53	udp	dns	0.00088	0	298	SHR	0	0	0	2	354	-	0	0	0	0	0	0	-	-	-	-	-	-	0	normal

Fig 1: Network Intrusion Detection Dataset

This dataset had a significant amount of data. Around 1,00,00,00 rows and 45 columns.

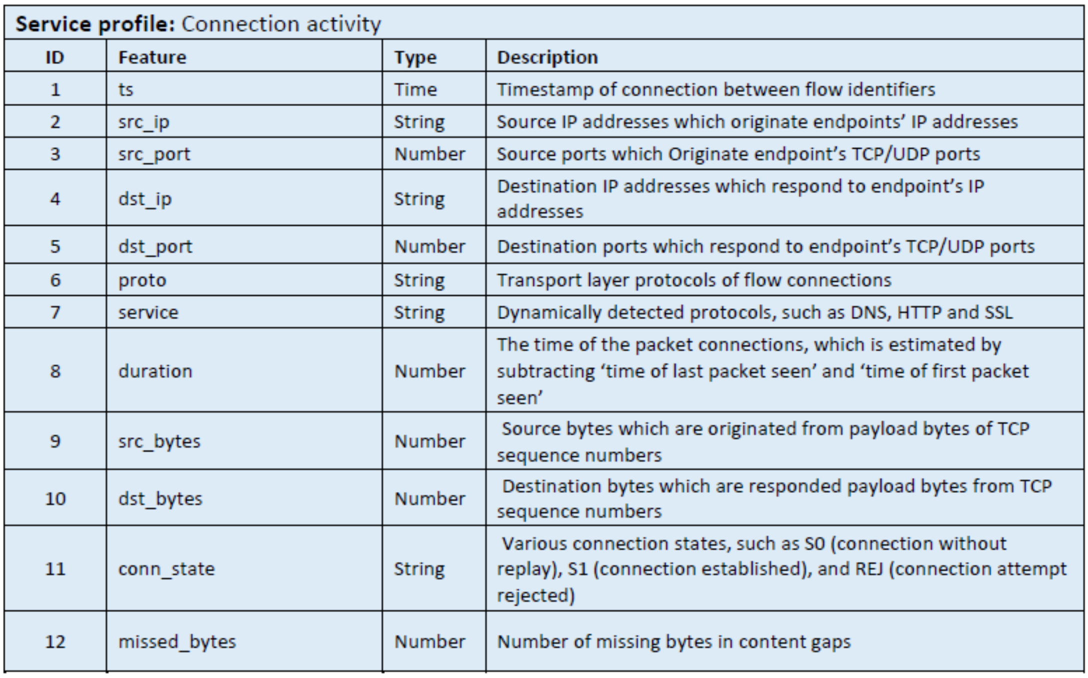


```
    #Viewing the shape and structure of our dataset/ counting rows and columns of the data set
    df.shape

(1000000, 45)
```

Fig 2: Rows and Columns of Network Intrusion Detection Dataset

The description of Network features is as follows:



Service profile: Connection activity			
ID	**Feature**	**Type**	**Description**
1	ts	Time	Timestamp of connection between flow identifiers
2	src_ip	String	Source IP addresses which originate endpoints' IP addresses
3	src_port	Number	Source ports which Originate endpoint's TCP/UDP ports
4	dst_ip	String	Destination IP addresses which respond to endpoint's IP addresses
5	dst_port	Number	Destination ports which respond to endpoint's TCP/UDP ports
6	proto	String	Transport layer protocols of flow connections
7	service	String	Dynamically detected protocols, such as DNS, HTTP and SSL
8	duration	Number	The time of the packet connections, which is estimated by subtracting 'time of last packet seen' and 'time of first packet seen'
9	src_bytes	Number	Source bytes which are originated from payload bytes of TCP sequence numbers
10	dst_bytes	Number	Destination bytes which are responded payload bytes from TCP sequence numbers
11	conn_state	String	Various connection states, such as S0 (connection without replay), S1 (connection established), and REJ (connection attempt rejected)
12	missed_bytes	Number	Number of missing bytes in content gaps

Fig 3: Network Intrusion Detection Dataset Features

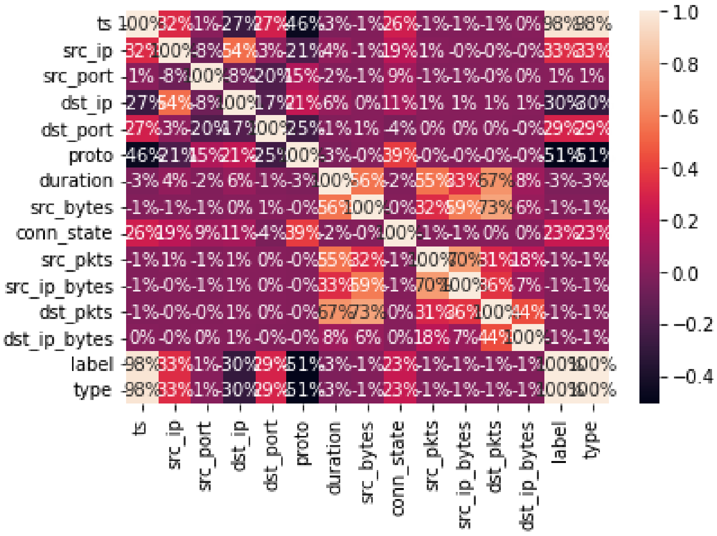
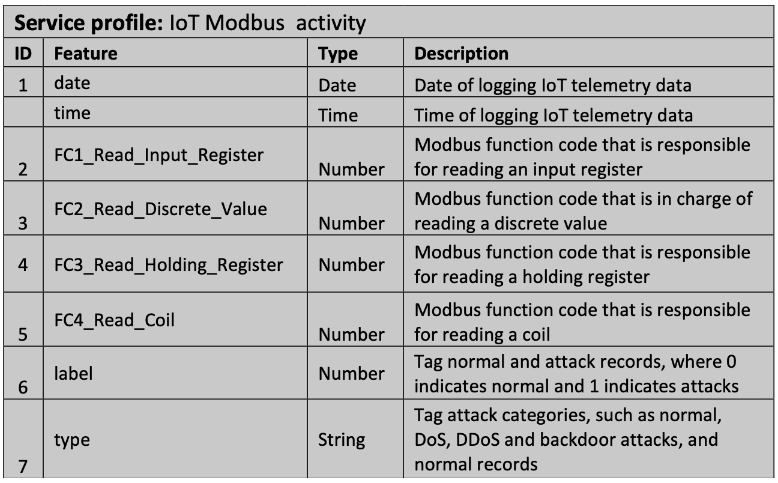


Fig 4: Heatmap of Network Intrusion Detection Dataset

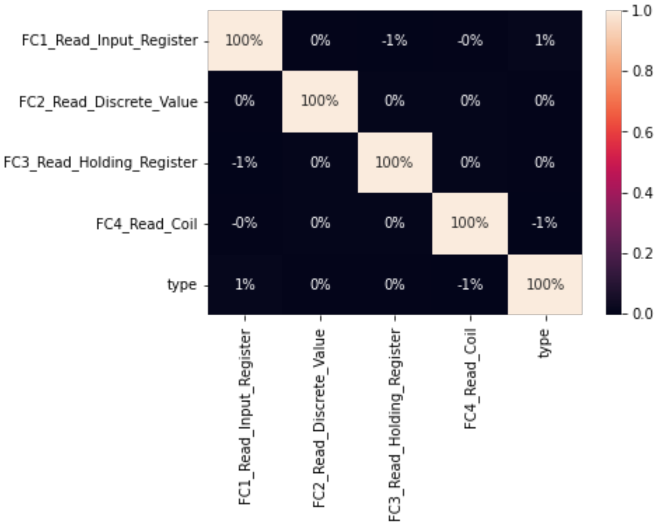
IoT Device Intrusion Detection: Several IoT devices and IoT sensors like Modbus were used to collect the telemetry data. In our instance, we have used the example of a Modbus to see if we can detect an intrusion or not. The dataset was relatively rich with data, containing 28,71,94 rows and 8 columns.
The features of the dataset are explained below -



Service profile: IoT Modbus activity

ID	Feature	Type	Description
1	date	Date	Date of logging IoT telemetry data
	time	Time	Time of logging IoT telemetry data
2	FC1_Read_Input_Register	Number	Modbus function code that is responsible for reading an input register
3	FC2_Read_Discrete_Value	Number	Modbus function code that is in charge of reading a discrete value
4	FC3_Read_Holding_Register	Number	Modbus function code that is responsible for reading a holding register
5	FC4_Read_Coil	Number	Modbus function code that is responsible for reading a coil
6	label	Number	Tag normal and attack records, where 0 indicates normal and 1 indicates attacks
7	type	String	Tag attack categories, such as normal, DoS, DDoS and backdoor attacks, and normal records

Fig 5: Features of IoT Device Intrusion Detection Dataset



	FC1_Read_Input_Register	FC2_Read_Discrete_Value	FC3_Read_Holding_Register	FC4_Read_Coil	type
FC1_Read_Input_Register	100%	0%	-1%	-0%	1%
FC2_Read_Discrete_Value	0%	100%	0%	0%	0%
FC3_Read_Holding_Register	-1%	0%	100%	0%	0%
FC4_Read_Coil	-0%	0%	0%	100%	-1%
type	1%	0%	0%	-1%	100%

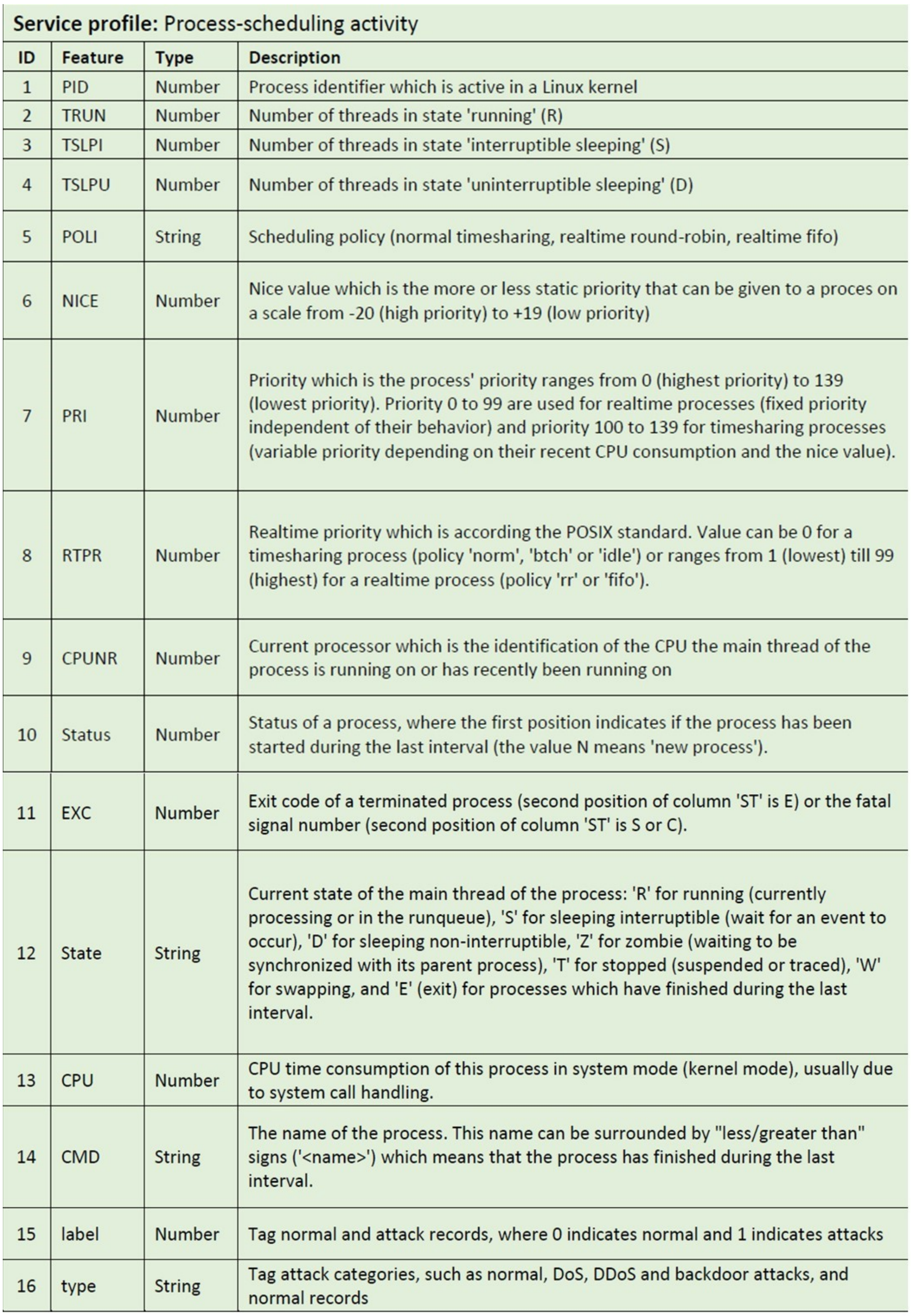
Fig 6: Heatmap of some select features of IoT Device Intrusion Detection Dataset

Linux (Ubuntu 14 18 Systems) Intrusion Detection:

Tracing tools like atop were run on Ubuntu 14 and 18 systems to record the usage of memory, disk, processor usage and network activities. In our instance, we used the dataset which listed the activity of the processor of the ubuntu machine.

The dataset was rich with plentiful data, having 1,00,00,00 rows and 17 columns.

Linux dataset features -



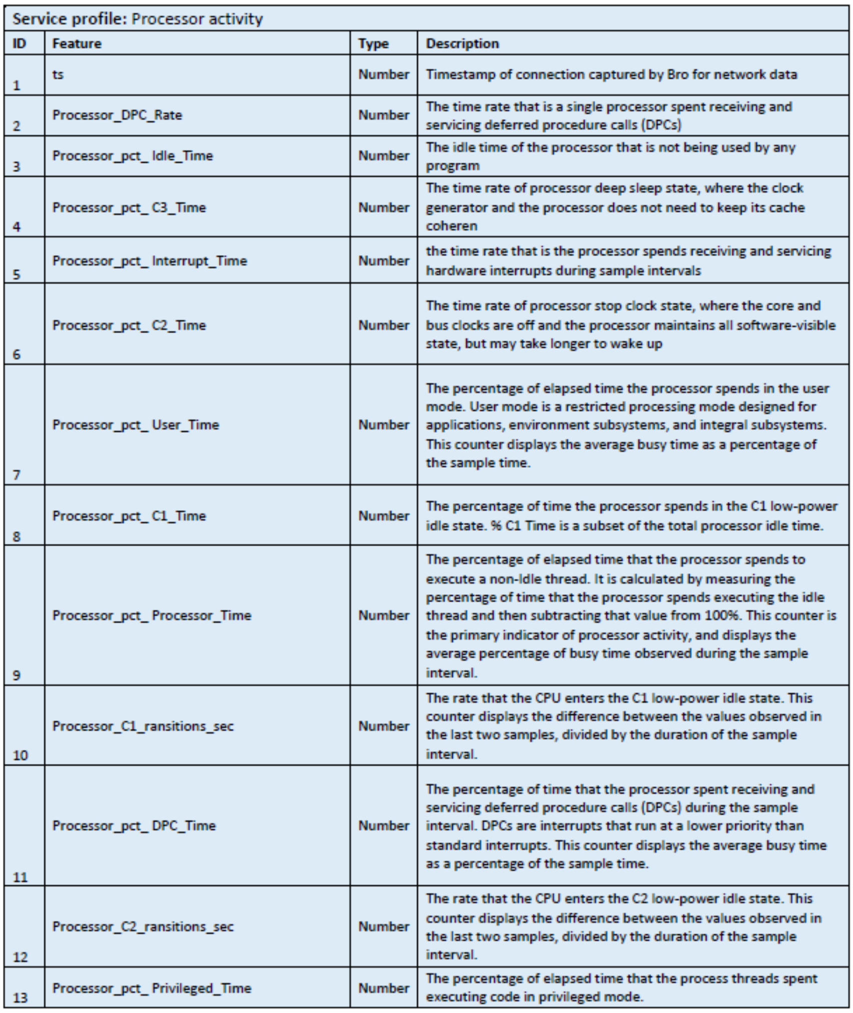
Service profile: Process-scheduling activity

ID	Feature	Type	Description
1	PID	Number	Process identifier which is active in a Linux kernel
2	TRUN	Number	Number of threads in state 'running' (R)
3	TSLPI	Number	Number of threads in state 'interruptible sleeping' (S)
4	TSLPU	Number	Number of threads in state 'uninterruptible sleeping' (D)
5	POLI	String	Scheduling policy (normal timesharing, realtime round-robin, realtime fifo)
6	NICE	Number	Nice value which is the more or less static priority that can be given to a proces on a scale from -20 (high priority) to +19 (low priority)
7	PRI	Number	Priority which is the process' priority ranges from 0 (highest priority) to 139 (lowest priority). Priority 0 to 99 are used for realtime processes (fixed priority independent of their behavior) and priority 100 to 139 for timesharing processes (variable priority depending on their recent CPU consumption and the nice value).
8	RTPR	Number	Realtime priority which is according the POSIX standard. Value can be 0 for a timesharing process (policy 'norm', 'btch' or 'idle') or ranges from 1 (lowest) till 99 (highest) for a realtime process (policy 'rr' or 'fifo').
9	CPUNR	Number	Current processor which is the identification of the CPU the main thread of the process is running on or has recently been running on
10	Status	Number	Status of a process, where the first position indicates if the process has been started during the last interval (the value N means 'new process').
11	EXC	Number	Exit code of a terminated process (second position of column 'ST' is E) or the fatal signal number (second position of column 'ST' is S or C).
12	State	String	Current state of the main thread of the process: 'R' for running (currently processing or in the runqueue), 'S' for sleeping interruptible (wait for an event to occur), 'D' for sleeping non-interruptible, 'Z' for zombie (waiting to be synchronized with its parent process), 'T' for stopped (suspended or traced), 'W' for swapping, and 'E' (exit) for processes which have finished during the last interval.
13	CPU	Number	CPU time consumption of this process in system mode (kernel mode), usually due to system call handling.
14	CMD	String	The name of the process. This name can be surrounded by "less/greater than" signs ('<name>') which means that the process has finished during the last interval.
15	label	Number	Tag normal and attack records, where 0 indicates normal and 1 indicates attacks
16	type	String	Tag attack categories, such as normal, DoS, DDoS and backdoor attacks, and normal records

Fig 7: Features of the Linux Dataset

Windows Intrusion Detection:
The Performance Monitoring Tool was used to trace the utilization of several resources like Memory, Disk, Ram, Processor and Network of Windows 10 machines. The dataset we chose to work with was extremely rich, bearing more than 125 features and over 35,000 rows of data.

This dataset had a lot of features. We tried to showcase a few of them:



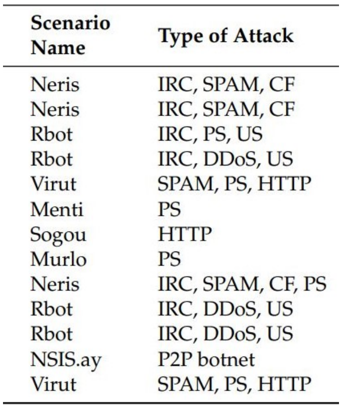
Service profile: Processor activity

ID	Feature	Type	Description
1	ts	Number	Timestamp of connection captured by Bro for network data
2	Processor_DPC_Rate	Number	The time rate that is a single processor spent receiving and servicing deferred procedure calls (DPCs)
3	Processor_pct_ Idle_Time	Number	The idle time of the processor that is not being used by any program
4	Processor_pct_ C3_Time	Number	The time rate of processor deep sleep state, where the clock generator and the processor does not need to keep its cache coheren
5	Processor_pct_ Interrupt_Time	Number	the time rate that is the processor spends receiving and servicing hardware interrupts during sample intervals
6	Processor_pct_ C2_Time	Number	The time rate of processor stop clock state, where the core and bus clocks are off and the processor maintains all software-visible state, but may take longer to wake up
7	Processor_pct_ User_Time	Number	The percentage of elapsed time the processor spends in the user mode. User mode is a restricted processing mode designed for applications, environment subsystems, and integral subsystems. This counter displays the average busy time as a percentage of the sample time.
8	Processor_pct_ C1_Time	Number	The percentage of time the processor spends in the C1 low-power idle state. % C1 Time is a subset of the total processor idle time.
9	Processor_pct_ Processor_Time	Number	The percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the percentage of time that the processor spends executing the idle thread and then subtracting that value from 100%. This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval.
10	Processor_C1_ransitions_sec	Number	The rate that the CPU enters the C1 low-power idle state. This counter displays the difference between the values observed in the last two samples, divided by the duration of the sample interval.
11	Processor_pct_ DPC_Time	Number	The percentage of time that the processor spent receiving and servicing deferred procedure calls (DPCs) during the sample interval. DPCs are interrupts that run at a lower priority than standard interrupts. This counter displays the average busy time as a percentage of the sample time.
12	Processor_C2_ransitions_sec	Number	The rate that the CPU enters the C2 low-power idle state. This counter displays the difference between the values observed in the last two samples, divided by the duration of the sample interval.
13	Processor_pct_ Privileged_Time	Number	The percentage of elapsed time that the process threads spent executing code in privileged mode.

Fig 8: Features of the Windows Dataset

3.2 The Hogzilla Dataset

We've settled on the Hogzilla Dataset, which was produced by fusing the network traffic behavioral traits from the CTU-13 Botnet and the ISCX 2012 IDS datasets. This labeled dataset includes 13 network traces of 7 different botnet malwares from the combined ISCX 2012 IDS dataset and CTU-13 botnet, together with information about genuine traffic, background traffic, and valid traffic's netflow metadata. The collection contains 7 botnets with the names Neris botnet, Rbot, Virut, Menti, Sogou, NSIS.ay, and Murlo. The botnets used IRC, SPAM, CF, PS, US, DDoS, HTTP, CF, and P2P botnets to conduct the attacks that were recorded in the dataset.[24]



Scenario Name	Type of Attack
Neris	IRC, SPAM, CF
Neris	IRC, SPAM, CF
Rbot	IRC, PS, US
Rbot	IRC, DDoS, US
Virut	SPAM, PS, HTTP
Menti	PS
Sogou	HTTP
Murlo	PS
Neris	IRC, SPAM, CF, PS
Rbot	IRC, DDoS, US
Rbot	IRC, DDoS, US
NSIS.ay	P2P botnet
Virut	SPAM, PS, HTTP

Fig 9: Name of Botnets and their attack type in CTU-13 dataset

192 subset of flow features are selected from the dataset which are sufficient enough to accurately detect the characteristics of the botnets. The features mainly consist of -

- Flow Duration
- Maximum and minimum expire time of the flow
- Protocol type
- Bytes count (Source to destination and vice-versa)
- Packets
- Source to destination packets
- Destination to source packets
- Maximum and minimum flow idle time

For running the dataset on our machine Learning models we preprocessed the dataset by removing all the noises and missing values, removed all unwanted rows and columns and transformed all the sting values to num values. This Feature Scaling is used in order to remove biases or deviance. The Min-Max Scaler method is used for this pre-processing. The dataset is divided into two sectors using the Train-Test-Split approach following the preparation stage, one for training and the other for testing. The splitting is done with a ratio which is gradually increased and decreased for better accuracy results of the machine learning models. The most efficient result was with a ratio of 75:25. Where 75% is for training and 25% is for testing. The training portion of the dataset is trained under the Machine Learning models. The Train-Test-Split method uses the X_train and y_train for training the models where X_train are the network flow features of the dataset and y_train is the Botnet itself. After the training is completed by the ML models, it is tested in the test portion of the dataset where X_test and y_test are the same features as the training ones. Each Machine learning model has a different execution time which is very distinctive from each other and mostly depending on the complexity of the models and the hardware it executes on.

IV. Proposed Model and Work Plan

4.1 Machine Learning

Machine Learning is a prominent division of AI (Artificial Intelligence), dedicated to building applications that work on enhancing precision by learning data with the flow of time, despite not being rigorously programmed to do so. It is a data-driven approach where data plays a fundamental role in constituting the overall accuracy of the program. There are mainly two forms of Machine Learning. They have Supervised Machine Learning and Unsupervised Machine Learning.

Machine Learning and it's Types

4.1.1 Supervised Machine Learning:

In this approach of Machine Learning, the output variable is defined, which means labeled data is fed to the models. The model is smart enough to associate mapping functions and map between input and output variables. Classification and Regression problems are examples of Supervised ML.

4.1.2 Unsupervised Machine Learning:

This technique of machine learning entails no labeling of data i.e. The model itself is smart enough to find the necessary patterns and structure embedded deep within the data. Association algorithms and clustering are examples of unsupervised ML. Machine Learning Models used in our Paper:

Linear Regression:

It is a form of Supervised ML technique. To put it simply, the primary objective of a linear regression model is to determine the linear connection between the independent and dependent variables. This is done by determining the best fit linear line between the variables [10]. In general, there are mainly two forms of linear regression, those being simple linear regression and Multiple linear regression. Simple linear regression is used when one independent variable is present. In this case, how the independent variable is related to the dependent variable and their linear relationship is determined by the simple linear regression. Whereas, in the case of multiple linear regression, several independent variables are used to identify relationships. The general equation of Linear Regression is:

$$y = b_0 + b_1 x$$

Here b_0 is the intercept. b_1 is the slope. In the equation, x is the independent variable and y is the dependent variable. As discussed earlier, the primary goal of a Linear Regression model is to minimise the error by finding the best-fit linear line, the ideal intercept and

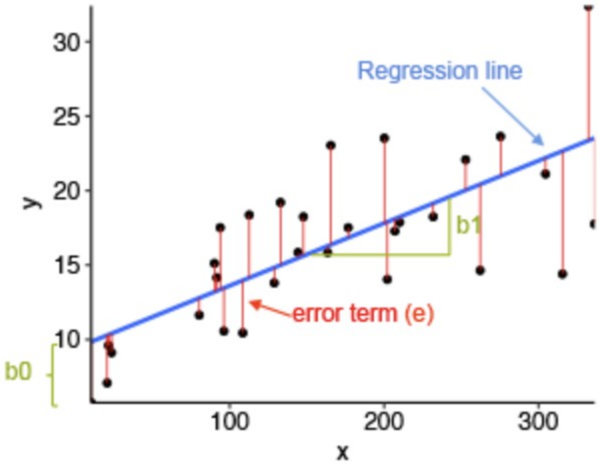


Fig 10: Linear Regression Visualization,[10]

coefficient values. Here, the error is basically the discrepancy between the predicted values and the actual values. The main objective is to minimize this discrepancy.
In the above figure, the dependent variable, y, is plotted along the y-axis and the independent variable, x, is plotted against the x-axis. The real values are the data points which are shown as black dots. Here, b1 is the slope of the x-line, and the intercept, b0, is equal to 10. The model's blue line, on which the projected values are located, is the line that fits the data the best. Error sometimes referred to as residual, is the vertical separation between the regression line and the data point. For every data point, one residual exists and the summation of all the differences constitutes the errors or sum of residuals.
This concept can be properly explained using an equation -

$$\sum e_i^2 = \sum (Y_i - \hat{Y}_i)^2$$

Postulations of Linear Regression-

- **Linearity:** The dependent variable Y should be linearly related to independent variables. We can verify this by plotting a scatter plot between the two variables.

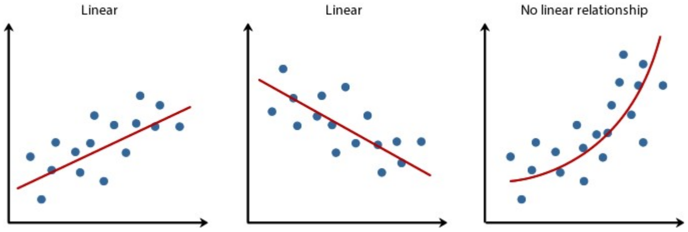


Fig 11: Linearity

- **Normality:** Variables X & Y should be normally distributed

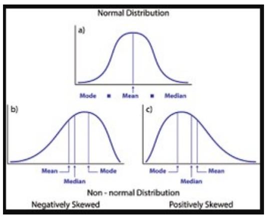


Fig 12: Normality

- **Homoscedasticity:** The spread of the residuals should be constant for each and every value of X i.e. The variance of the error terms should be constant. We can verify this by plotting a residual plot. If our residuals are correct, they will form a constant variance, however, if they are not, a funnel-shaped residual will be created.

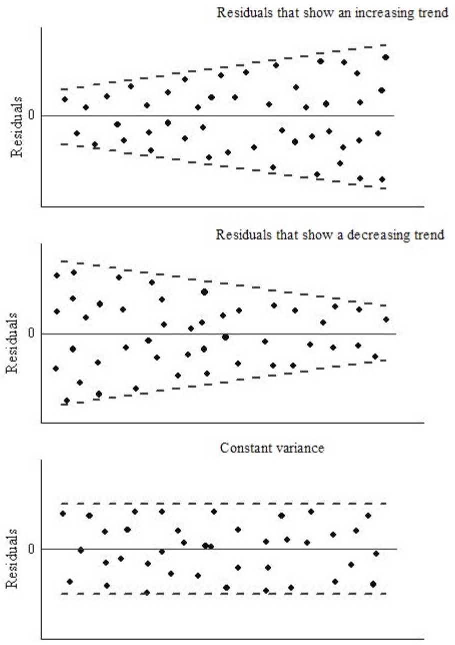


Fig 13: Homoscedasticity

Logistic Regression:

The central component of Logistic Regression is the Logistic Function and this inspired the name of this model or technique. The Sigmoid Function or the Logistic Function was created by experts in Statistics in order to distinguish the features of population augmentation in ecology, which escalate promptly and peak at the carrying capacity of the ecosystem. Using the S-shaped curve, any number in the domain of the real realm can be recast between 0 and 1. However, they can never be precisely at those ranges. Let us see an instance where we plot a logistic transformation of the numbers between -5 and 5 into the range of 0 and 1. In this case, e is the base of the natural logarithm and the value is the actual numerical value we want to alter. The logistic transformation is plotted below.

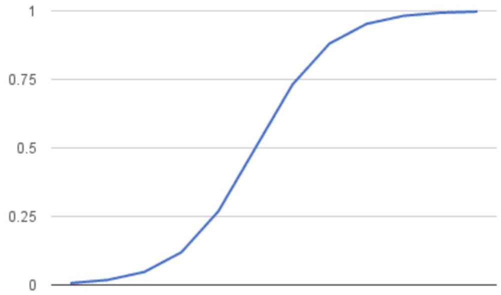


Fig 14: Plotting of Logistic Function Curve, [17]

Logistic regression, similar to Linear Regression, represents data using an equation. Input values (x) are incorporated with coefficient values or weights (y) (commonly referred to as Beta which is a Greek Capital Letter) in order to predict the output values. One significant difference between Logistic Regression from Linear Regression is that, instead of modelling the output value in terms of numeric values, the output values are simply modelled using binary values (0 or 1). Y is the expected result (x) when the single input value coefficient is b1 and the intercept. From our training set, we must learn the corresponding b coefficients (constant real values) for each column in our input data.
Below is an example logistic regression equation: $y = e^{(b_0+b_1*x)/(1+e^{(}b_0+b_1*x))}$.
Logistic regression has one crucial benefit, that is, it enables us to examine numerous explanatory variables by expanding the rudimentary ideas. The basic formula:

$$P = \frac{1}{1+e^{-(\beta_0+\beta_1X_1+\beta_2X_2+\ldots\beta_nX_n)}} = \frac{1}{1+e^{-(\beta_0+\sum\beta_iX_i)}}.$$

Fig 15: Logistic regression formula, [15]

Here, the slope (β_i) determines how steep the curve is and the constant (β_0) shifts the curve left and right. The Maximum Likelihood Estimation or MLE is the most prominent method for gauging the beta parameter, or coefficient, in this model.

Decision Tree

Amongst the Supervised Machine Learning algorithms, the Decision Tree stands as a quintessential instance. One of the most prominent, efficient and preferred techniques for prediction and categorization is the decision tree Here, the data is bifurcated in a continual manner following certain parameters. In Decision Trees, the tree can be easily explained using two entities i.e. leaves and nodes. The structure of the decision tree resembles a flowchart. Here, every leaf constitutes the class label or node, every internal node stands for a test on an attribute and every branch signifies an output of the test result.

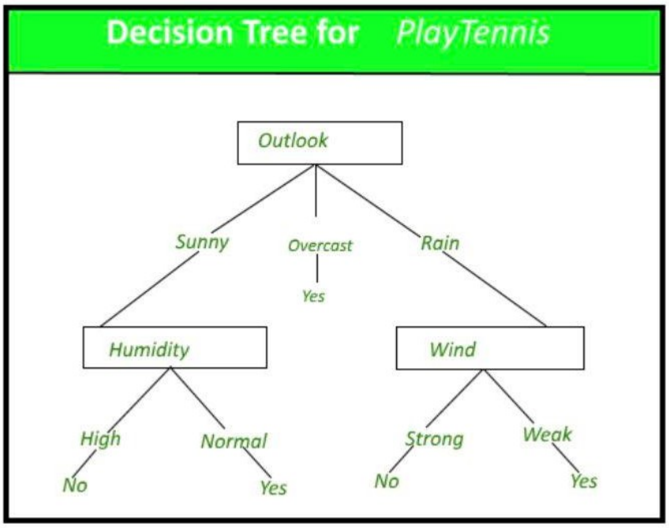


Fig 16: Example of Decision Tree,[12]

Building a Decision Tree: A tree can be easily "learned" by promptly dividing the source-set into sub-groups based on an attribute value test. It is known as recursive partitioning to repeat this operation on each derived subset. In this process, every feature is taken into account, and several split points are investigated and evaluated using a cost function. The recursion is finished when the subset at a node bears the same value for the target variable or when the split no longer enhances the predictions. The division with the least cost is chosen. Decision tree classifier building is ideal for exploratory knowledge discovery because it doesn't require the configuration of a parameter or understanding of a domain. Decision Trees can also handle High-dimensional data. Decision tree classifiers are often accurate. Decision tree induction is a popular inductive method for learning classification information.
Here, the decision trees learn from data to approximate a sine curve using a series of if-then-else decision rules. If the tree is deep, the decision criteria become more complex and the model is more accurate. The target value is along the Y-axis while the features of the dataset are along the X-axis. The concept of a Decision Tree Regression is similar to a Decision Tree Classifier where we use a binary tree until we are left with a pure leaf node. The dataset is split on the basis of the condition of the root node. The splitting points are denoted in the graph by the blue and green lines. This splitting rule removes the impurity of the nodes by following the condition and ignoring the values that sway away from the pattern. The best condition is found through repeated analysis of data variance. A higher value of variance denotes higher impurity. The decision tree results in a sine wave of the points that adhere the most to the best condition.

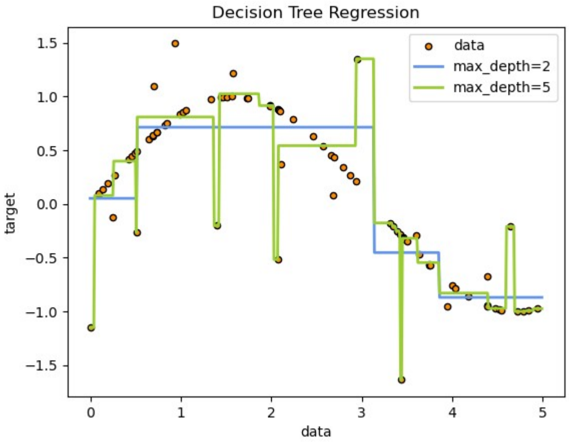


Fig 17: Decision Tree Regression, [1]

$$Variance = \frac{\sum (X-\mu)^2}{N}$$

Random Forest Classifier

Multiple separate decision trees that collaborate as an ensemble make form a random forest. After many decision tree classifiers have been fitted to various dataset subsamples, this meta estimator uses averaging to improve predicted accuracy and decrease overfitting. Each tree is built using the entire dataset if bootstrap=True (the default), in which case the size of the sub-sample is defined by the max samples argument. [29]

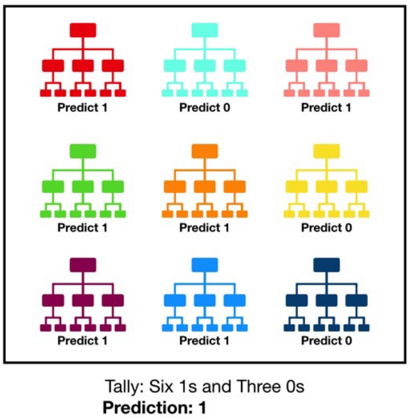


Fig 18: Random Forest Classifier

Randomness in feature selection – In a traditional decision tree, while splitting a node, we examine all possible features and select the one that produces the biggest divergence between the observations in the left node and those in the right node. Each tree in a random forest can only access a random subset of traits. By driving even more variance throughout the model's trees, this ultimately results in less tree correlation and more diversity.
In order for random forest to function properly,

- In order for models created utilizing those attributes to perform better than guesswork, there must be some real signal in those features.

- Low correlations between the predictions (and thus the mistakes) of the separate trees are required.

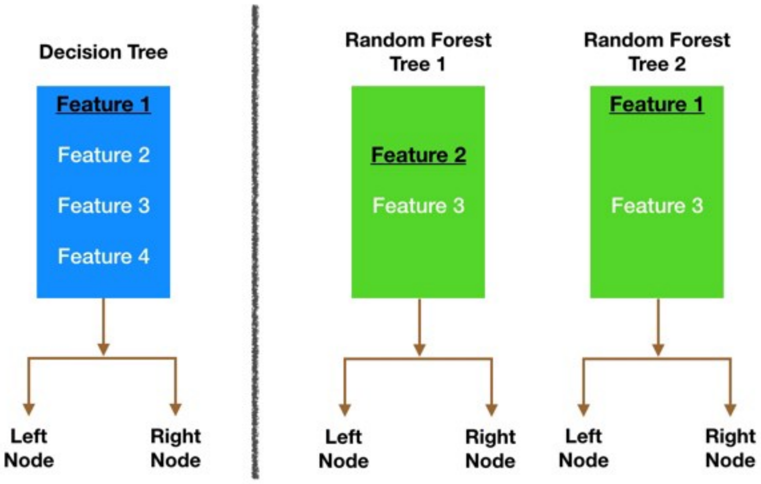


Fig 19: Decision tree and Random Forest Classifier comparison

In the image above, the typical decision tree (in blue) may choose from any of the four attributes to choose how to split the node. Let's walk through an example using visuals. As it divides the data into groups that are as distinct as possible, it chooses to use Feature 1 (black and underlined).

In random forest, when we examine Random Forest Tree 1, we see that it can only take into account the randomly chosen Features 2 and 3. The optimal feature for splitting, according to our conventional decision tree (in blue), is feature 1, but because Tree 1 cannot see feature 1, it must choose feature 2. (black and underlined). In contrast, Tree 2 can only view Features 1 and 3, hence it is able to pick feature 1.

It can be summarized into four stages:

- Select random samples from the specified dataset.
- For each sample, make a decision tree, and then examine the predictions it yields.
- Vote for each anticipated result.
- The result with the most votes should be the final prediction.

One of the key advantages of random forest is its versatility. It allows for easy visualization of the relative weights it assigns to the input characteristics and may be utilized for both classification and regression problems.

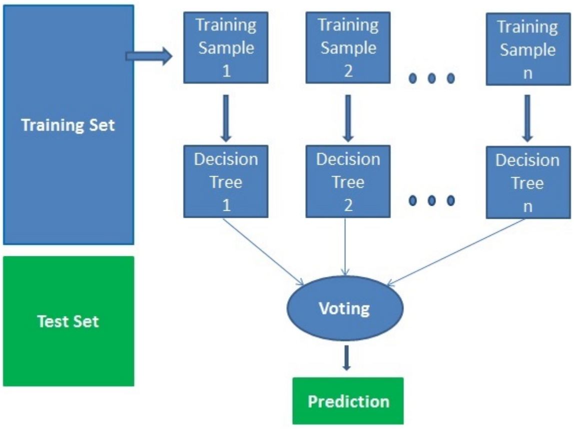


Fig 20: Random Forest Classifier Train_Test_Split

(NNC) Neural Network Classifier

Neural networks are largely influenced by the human brain's learning process. They are composed of an artificial network of parameters that the computer may analyze in order to learn from and fine-tune. These variables are frequently referred to as a "neuron," which receives one or more inputs and produces a result. These outputs are received by the layer of neurons below, who then use them as inputs to carry out their own tasks and generate more outputs. The output from the end neurons is sent to the next layer of neurons once each layer of neurons has been considered and the input has reached them. The final output for the model is then produced by those final neurons. The algorithm on which the neural network classifier executes is known as the MLP or Multi-layer Perceptron. [19]

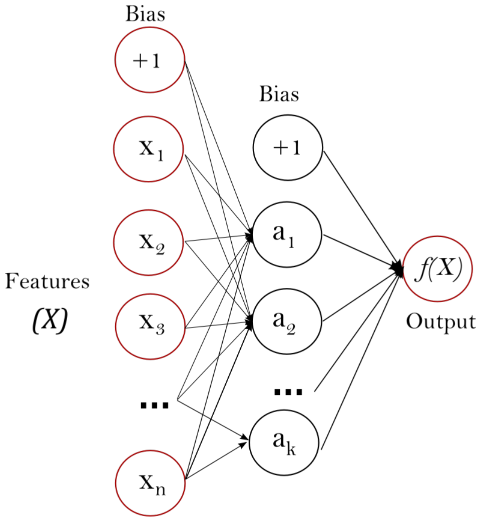


Fig 21: Neural Network Classifier

Classification Types for NNC

- For predicting with two possible outcomes, Binary Classification is used.

- for predicting within more than two possible outcome, Multi-class Classification is used.
- For predicting within which categories should be assigned to a particular feature, Multi-label Classification is used mostly because one feature could have more than one category assigned.

MLP is a learning algorithm also known as Multi-layer Perceptron. An artificial neural network model called the multi-Layer perceptron or MLP transforms data inputs into a series of useful outputs. A MLP is made up of several layers, which are all completely linked to the previous one before. The binary Classification and Multi-Class Classification is done using MLPCLassifier. There are a few advantages of using MLP which are mentioned below:

- It can learn nonlinear models.
- by using the partial_fit it can learn models in real-time.

4.2 Hybrid Quantum Machine Learning Model

We plan to compare our results generated from the classical machine learning models above to a hybrid quantum model. The quantum model we plan to use will take classical data(data represented in bits) as input and process that data in a quantum machine.

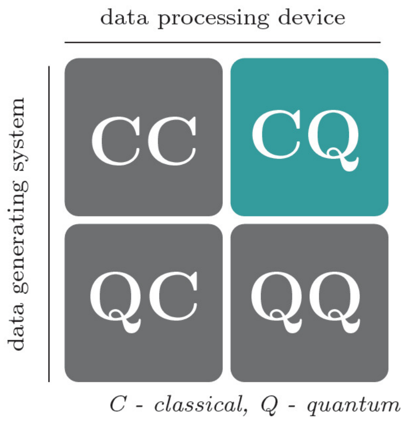


Fig 22: Classical Data Processed Using a Quantum Machine

To process the classical input in a quantum fashion, we will need to use a quantum model. Deterministic quantum models and Variational quantum models are the two broad categories into which quantum models can be divided.[5]
In deterministic quantum models, we can create an algorithm or model where, when we measure our quantum system, we know with certainty what the outcome is going to be. One such example is the Deutsch-Jozsa algorithm.[5]

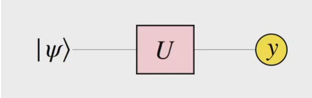


Fig 23: Deterministic Quantum Model Workflow

Here, the first parameter is a ket-vector(complex column vector) undergoing an operation(U). This operation can be something like a matrix multiplication or some other matrix function. Finally 'y' is the result of the measurement of the system.
But, the downside of these models is that they tend to be quite noisy. Which means they are susceptible to interference and hence will sometimes produce outputs that are not accurate.[5]

Variational quantum models are the newer generation of quantum models. Variational quantum models differ from their differential counterparts in the sense that the output generated is not deterministic, rather the output is stochastic. So, these models are run many times to get statistics of the output (expectation value). The variational quantum models are more similar to many classical machine learning models and include, quantum support vector machines and quantum neural networks and a large portion of quantum machine learning models fall into this category.[5]

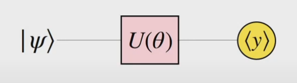


Fig 24: Variational Quantum Model Workflow

The working principle is very similar to that of the deterministic model, but with some key differences. These variational models depend on some parameters(θ) we can tweak and change. So the operations, U, are parameterized, while '$< y >$' is the stochastic output of the model.
What makes these variational quantum models better is that they are less susceptible to noise and are less likely to generate wrong outputs.[5]

The goal is to use a quantum simulator rather than a quantum machine as getting access to run a procedure on such a machine requires queuing a job in the IBM quantum machine servers. Furthermore, modern quantum computers are quite small and noisy. Hence, every test that is run will have a random amount of noise associated with it. In addition to that, since the quantum computers are used by other people too, one has to queue up and the wait times may vary. Even after gaining access to the quantum computer, there is a maximum allotted time before someone else gets access again. Taking into account all these factors, we have opted to use a simulator which will give us a noiseless result and simulate an ideal case or near ideal case for the future, when quantum computers have improved a great deal.

Our proposed model aims to detect and as a result prevent IoT botnets from infecting vulnerable IoT devices. Although there are multiple methods to detect botnets, we think that ours will yield better results by efficiently making the model learn the training and testing datasets so that the model becomes more adept at recognizing potential intruding botnets. To accomplish so, the model necessitates the creation of a mechanism that accepts data from IoT devices as an input, processes it systematically, and produces predictions of two types: ”regular” and ”irregular.” To achieve this, we have a roughly sketched blueprint of how we plan to implement our model. To elaborate:

1. Pick a suitable dataset.

2. Feature Selection for finding out useful data to test and train on.

3. Cluster and label the dataset based on the selected features and sort out the outliers.
4. Prepare a machine learning model and a quantum machine learning model based on the original machine learning model.
5. Feed the specific features to each of our models.
6. Test our trained models against different random datasets.
7. Compare and contrast the results of each model.
8. Refine the Quantum model if necessary to achieve optimum results.

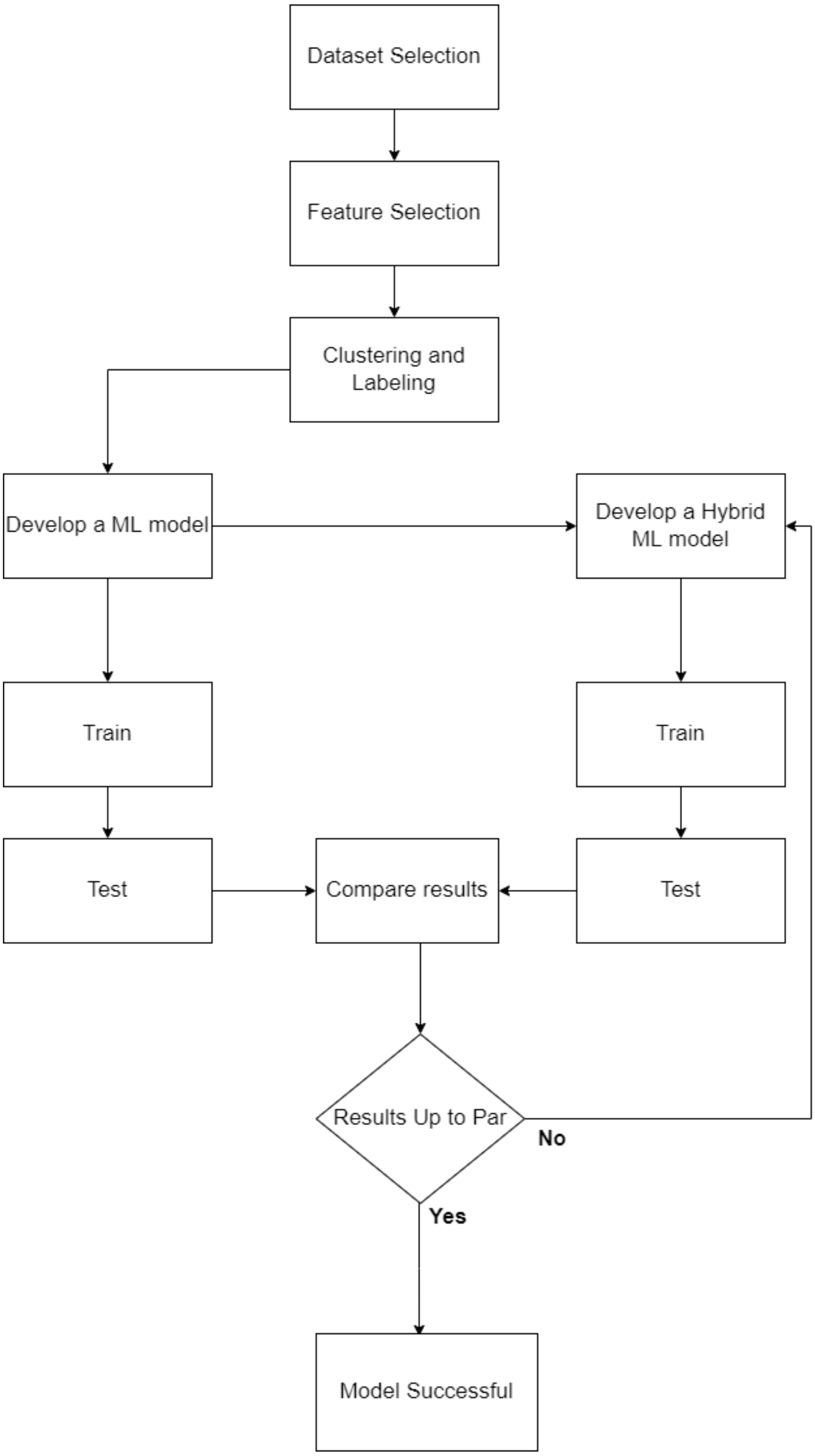


Fig 25: Flowchart representing the workflow for IoT Bot detection

V. Preliminary analysis

The Result of Implementing ML in the dataset of Network Intrusion Detection

We followed the steps mentioned and found the following results -
Accuracy of Logistic Regression: 0.847065
Time Required for Logistic Regression: 6.4s
Accuracy of Decision Tree: 0.999995
Time Required for Decision Tree: 1.6s
NNC Training accuracy is: 0.976348750
NNC Testing accuracy is: 0.976230000
Time Required for NNC: 1m 47.5s

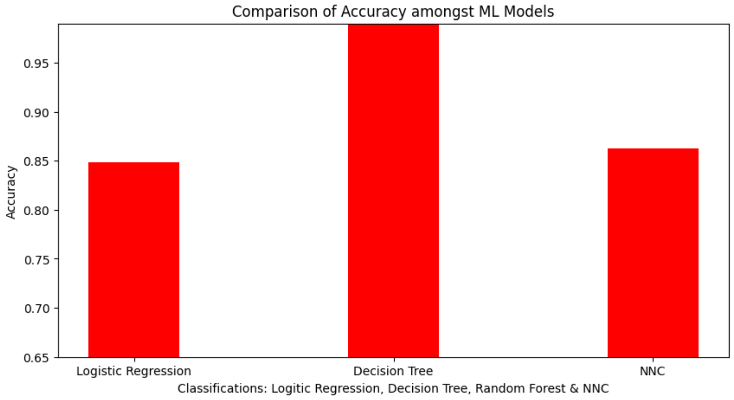


Fig 26: Accuracy Comparison in Network Intrusion Detection

The Result of Implementing ML in the dataset of IoT Device Intrusion Detection

After running the aforementioned ML models, we found the following:
Accuracy of Logistic Regression: 0.7768415188286704
Time Required - 5.7s
Accuracy of Decision Tree: 0.9466912724803704
Time Required - 46.6s
Training accuracy of Random Forest is 0.995064308
Testing accuracy of Random Forest is 0.966608054
Time Required - 46.6s
Training accuracy of NNC is 0.775756784
Testing accuracy of NNC is 0.776841519
Time Required - 1m 25.6s

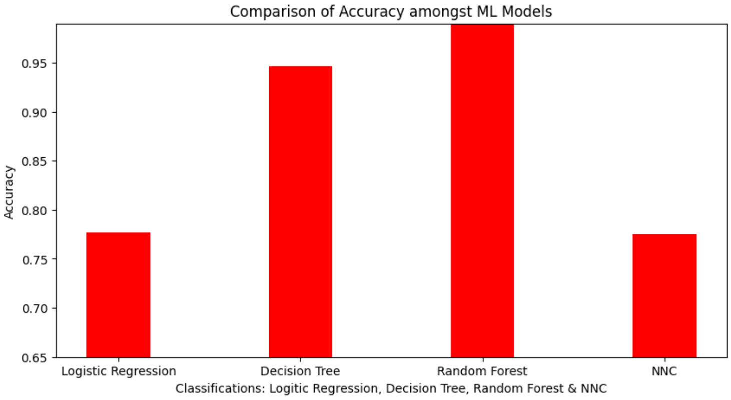


Fig 27: Accuracy Comparison in IoT Device Intrusion Detection

The Result of Implementing ML in the dataset of Linux (Ubuntu 14 18 Systems) Intrusion Detection

We followed the aforementioned steps and found the following results:
Accuracy of Logistic Regression: 0.87823
Time Required: 40.2s
Accuracy of Decision Tree: 0.890845
Time Required: 2.3s
Training accuracy of Random Forest is: 0.898576250
Testing accuracy of Random Forest is: is 0.890730000
Time Required: 28.2s
Training accuracy of NNC is: 0.879585000
Testing accuracy of NNC is: 0.878275000
Time Required: 2m 3.6s

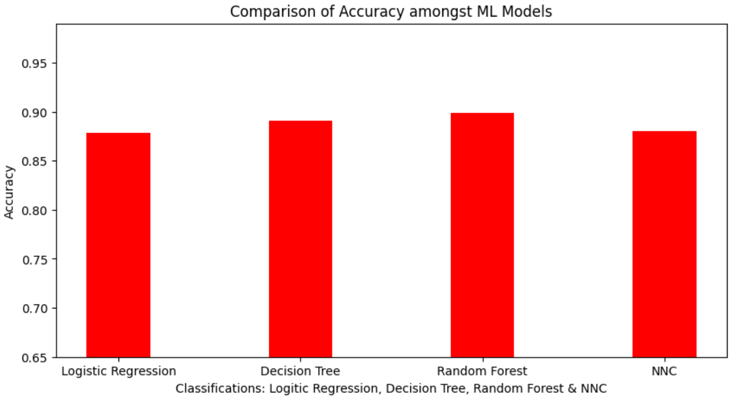


Fig 28: Accuracy Comparison in Linux Intrusion Detection

The Result of Implementing ML in the dataset of Windows Intrusion Detection

We followed the aforementioned steps and got the following results:
Accuracy of Logistic Regression: 0.9570535093815149
Time Required: 2.2s
Accuracy of Decision Tree: 0.977901320361362

Time Required: 3.8s
Training accuracy of NNC is: 0.692564281
Testing accuracy of NNC is: 0.686587908
Time Required: 7.2s
Training accuracy of Random Forest is: 0.999548297
Testing accuracy of Random Forest is: 0.984433634
Time Required: 3.8s

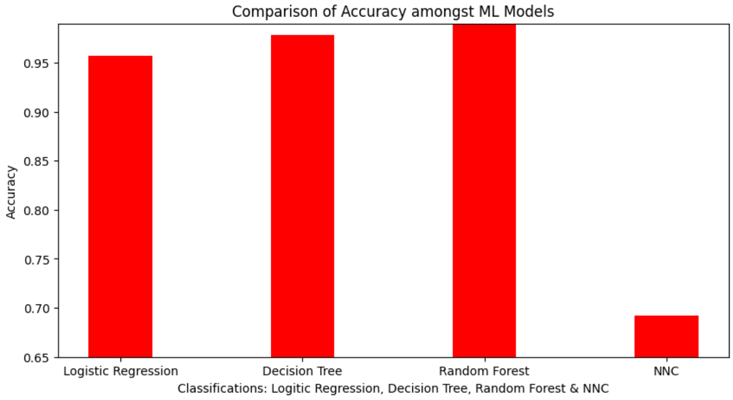


Fig 29: Accuracy Comparison in Windows Intrusion Detection

The Result of Implementing ML in Hogzilla Dataset

Results The accuracy and execution-time results of the ML models are stated below:
Accuracy of Linear Regression : 62.64%
execution time: 16.32 sec
Accuracy of Neural Network Classifier : 71.97 %
execution time: 435.51 sec
Accuracy of Logistic Regression : 78.52 %
execution time: 141.3 sec
Accuracy of Decision Tree : 96.61%
execution time: 23.92 sec
Accuracy of Random Forest Classifier : 96.81 %
execution time: 85.43 sec

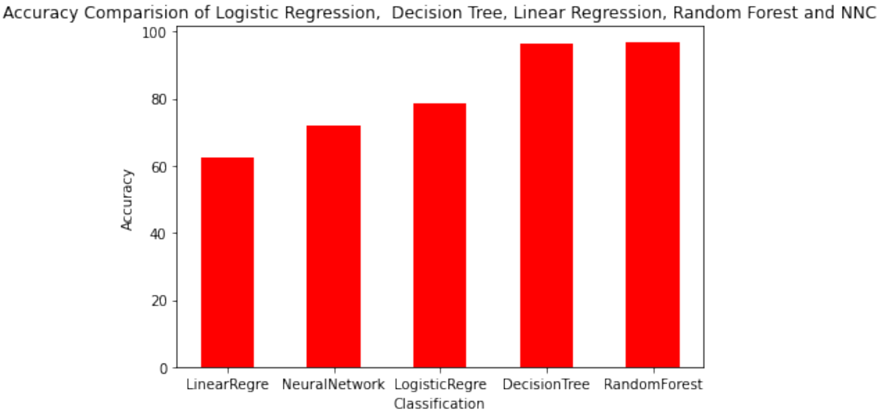


Fig 30: Accuracy Comparison in Hogzilla

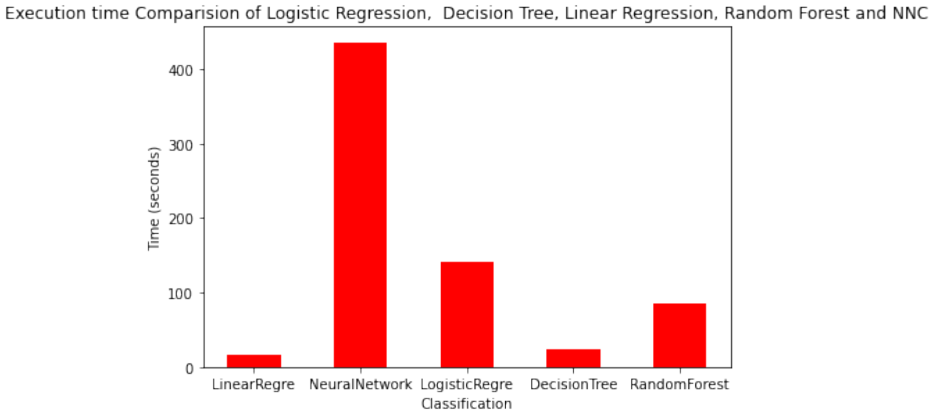


Fig 31: Execution Time Comparison in Hogzilla

By examining the accuracy ratings, we can see that the Neural Network Classifier has an accuracy rate that is 9% more than the Linear regression model, whereas the Logistic Regression model has 6.4% more accuracy than the Neural Network Classifier. The Decision tree Classifier and Random Forest Classifier models have the most detection accuracy which is slightly above 96%. The comparison is visually represented in the bar chart below. By comparing the execution times, the Neural network classifier has the highest execution time because of the number of features the dataset has. Whereas, linear regression method has the least execution time. It is mostly due to the number of features. The decision tree machine learning method gave the most accuracy with the least amount of time.

VI. Conclusion

The existing methodologies used for the detection of advanced IoT botnets, although reasonably sustainable, are not entirely efficient. In our study, we concentrated on ToN IoT and Hogzilla, two of the most well-documented datasets on IoT, attempted to train them using the models mentioned above. The two classifiers that produced the highest accurate results overall were Random Forest Classifier and Decision Tree Classifier. Despite their remarkable precision, these take too long to execute. If at all possible, we want to maximize accuracy while also drastically cutting down on execution time. We believe that implementing a quantum hybrid layer will help us accomplish our goals more quickly and efficiently.

References

[1] *1.10. Decision Trees*. URL: https://scikit-learn.org/stable/modules/tree.html.

[2] Florian Adamsky et al. “Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver”. In: *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 2012, pp. 143–150. DOI: 10.1109/CyberC.2012.31.

[3] Radware ERT Threat Advisory. *Hajime – friend or foe?* 2017. URL: https://security.radware.com/ddos-threats-attacks/hajime-iot-botnet/.

[4] Francisco Villegas Alejandre, Nareli Cruz Cortés, and Eleazar Aguirre Anaya. “Feature selection to detect botnets using machine learning algorithms”. In: *2017 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. IEEE. 2017, pp. 1–7.

[5] Alexandria Quantum Computing Group. *An Introduction to Quantum Machine Learning*. Feb. 2021. URL: https://www.youtube.com/watch?v=yHG2z_BQJ8M&t=882s%5C&ab_channel=AlexandriaQuantumComputingGroup.

[6] Anna-senpai. *Mirai source code*. 2017. URL: https://github.com/jgamblin/Mirai-Source-Code/..

[7] Manos Antonakakis et al. “Understanding the mirai botnet”. In: *26th USENIX security symposium (USENIX Security 17)*. 2017, pp. 1093–1110.

[8] Yagnesh Balasubramanian et al. “Quantum IDS for mitigation of DDoS attacks by mirai botnets”. In: *International Conference on Next Generation Computing Technologies*. Springer. 2017, pp. 488–501.

[9] Agathe Blaise et al. “Botnet fingerprinting: A frequency distributions scheme for lightweight bot detection”. In: *IEEE Transactions on Network and Service Management* 17.3 (2020), pp. 1701–1714.

[10] Deepanshi. *All you need to know about your first Machine Learning model – Linear Regression*. May 2021. URL: https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/.

[11] Madhuri Gurunathrao Desai, Yong Shi, and Kun Suo. “A Hybrid Approach for IoT Botnet Attack Detection”. In: *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2021, pp. 0590–0592.

[12] GeeksforGeeks. *Decision Tree*. Aug. 2022. URL: https://www.geeksforgeeks.org/decision-tree/.

[13] Shreehar Joshi and Eman Abdelfattah. “Efficiency of Different Machine Learning Algorithms on the Multivariate Classification of IoT Botnet Attacks”. In: *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE. 2020, pp. 0517–0521.

[14] G Kirubavathi and R Anitha. “Structural analysis and detection of android botnets using machine learning techniques”. In: *International Journal of Information Security* 17.2 (2018), pp. 153–167.

[15] George Lawton, Ed Burns, and Linda Rosencrance. *logistic regression*. Jan. 2022. URL: https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression.

[16] Vailshery Lionel. *Global IoT and Non-IoT Connections 2010-2025*. 2021, Mar 8. URL: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[17] *Logistic Regression*. URL: http://faculty.cas.usf.edu/mbrannick/regression/Logistic.html.

[18] Christopher D McDermott, Farzan Majdani, and Andrei V Petrovski. “Botnet detection in the internet of things using deep learning approaches”. In: *2018 international joint conference on neural networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[19] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[20] Eyal Ronen et al. “IoT goes nuclear: Creating a ZigBee chain reaction”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 195–212.

[21] Weam Saadi et al. “IoT Botnet Detection: Challenges and Issues”. In: *Test Engineering and Management* 83 (June 2020), pp. 15092–15097.

[22] Hatma Suryotrisongko and Yasuo Musashi. “Evaluating hybrid quantum-classical deep learning for cybersecurity botnet DGA detection”. In: *Procedia Computer Science* 197 (2022), pp. 223–229.

[23] Hatma Suryotrisongko and Yasuo Musashi. “Hybrid Quantum Deep Learning with Differential Privacy for Botnet DGA Detection”. In: *2021 13th International Conference on Information & Communication Technology and System (ICTS)*. IEEE. 2021, pp. 68–72.

[24] *The hogzilla dataset · Hogzilla Ids*. Jan. 2016. URL: https://ids-hogzilla.org/dataset/.

[25] *The* $TON_{I}oTDatasets|UNSWResearch$. URL: https://research.unsw.edu.au/projects/toniot-datasets.

[26] Knecht Tobias. *A brief history of bots and how they’ve shaped the internet today*. 2021, May 5. URL: https://abusix.com/resources/botnets/a-brief-history-of-bots-and-how-theyve-shaped-the-internet-today/.

[27] Tong Anh Tuan et al. “Performance evaluation of Botnet DDoS attack detection using machine learning”. In: *Evolutionary Intelligence* 13.2 (2020), pp. 283–294.

[28] Shingo Yamaguchi. “Botnet Defense System: Concept and Basic Strategy”. In: *2020 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2020, pp. 1–5.

[29] Tony Yiu. *Understanding random forest*. Sept. 2021. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[30] Omer Yoachimik. *Mantis - the most powerful botnet to date*. Aug. 2022. URL: https://blog.cloudflare.com/mantis-botnet/.