NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MACHINE LEARNING FOR MALWARE BOTNET DETECTION IN IOT DEVICES

by

Charles R. Gallagher

June 2021

Thesis Advisor: Robert A. Koyak
Second Reader: Jarrod S. Shingleton

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2021	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE MACHINE LEARNING FOR MALWARE BOTNET DETECTION IN IOT DEVICES	5. FUNDING NUMBERS
6. AUTHOR(S) Charles R. Gallagher	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

Cyber threats against the Department of Defense (DOD) and the greater U.S. public create an ever-increasing security challenge. Advances in information technology provide new capabilities and benefits but also vulnerabilities. Today, the internet of things (IoT) is almost everywhere. Homes, businesses, and government organizations are continuing to add internet-connected devices for increased productivity and convenience. Military IoT devices provide traditional computing as well as specific functional purpose sensors. The DOD will increasingly depend upon a diverse range of IoT devices to gain information dominance over its adversaries. IoT technology in real time can provide entity-level maintenance, logistics, and intelligence data that has the potential to enable command and control decisions with greater confidence and speed. However, IoT devices are vulnerable to attack by malware, which has proven to be a network security concern. There have been many high-profile attacks such as the Mirai botnet and SolarWinds breaches that demonstrate IoT vulnerabilities. Advances in machine learning offer potential solutions to detect the evolving nature of cyber intrusions on internet networks. This thesis examines approaches to detecting malware-infected devices using machine learning and labeled IoT network flow data. It also seeks to determine whether supervised machine-learning models provide generalizable solutions for malware detection on new networks and IoT devices.

14. SUBJECT TERMS machine learning, Zeek, cyber security, IoT, deep learning, LSTM, network security	15. NUMBER OF PAGES 99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MACHINE LEARNING FOR MALWARE BOTNET DETECTION IN IOT DEVICES

Charles R. Gallagher
Major, United States Army
BA, Virginia Polytechnic Institute and State University, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Robert A. Koyak
Advisor

Jarrod S. Shingleton
Second Reader

W. Matthew Carlyle
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Cyber threats against the Department of Defense (DOD) and the greater U.S. public create an ever-increasing security challenge. Advances in information technology provide new capabilities and benefits but also vulnerabilities. Today, the internet of things (IoT) is almost everywhere. Homes, businesses, and government organizations are continuing to add internet-connected devices for increased productivity and convenience. Military IoT devices provide traditional computing as well as specific functional purpose sensors. The DOD will increasingly depend upon a diverse range of IoT devices to gain information dominance over its adversaries. IoT technology in real time can provide entity-level maintenance, logistics, and intelligence data that has the potential to enable command and control decisions with greater confidence and speed. However, IoT devices are vulnerable to attack by malware, which has proven to be a network security concern. There have been many high-profile attacks such as the Mirai botnet and SolarWinds breaches that demonstrate IoT vulnerabilities. Advances in machine learning offer potential solutions to detect the evolving nature of cyber intrusions on internet networks. This thesis examines approaches to detecting malware-infected devices using machine learning and labeled IoT network flow data. It also seeks to determine whether supervised machine-learning models provide generalizable solutions for malware detection on new networks and IoT devices.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

List of Acronyms and Abbreviations

API	Application Programming Interface
C&C	Command & Control
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance
CPU	Central Processing Unit
CTU	Czech Technical University
CSV	Comma Separated Values
DARPA	Defense Advanced Research Projects Agency
DT	Decision Tree
DNN	Deep Neural Network
DNS	Domain Name System
DSAE	Deep Sparse AutoEncoder
DoD	Department of Defense
DoS	Denial of Service
DDoS	Distributed Denial of Service
FPR	False Positive Rate
FTP	File Transfer Protocol
GB	Gigabyte
GPU	Graphics Processing Unit
HPC	High Performance Computer

HTTP	Hypertext Transfer Protocol
HTTPs	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRC	Internet Relay Chat
KDD	Knowledge Discovery and Data Mining
LITNET	Lithuanian Research and Education Network
LSTM	long Short-Term Memory
MCC	Matthew's Correlation Coefficients
MIoT	Military Internet of Things
NPS	Naval Postgraduate School
NETCOM	Network Enterprise Technology Command
PCAP	Packet Capture
POP3	Post Office Protocol version 3
ReLu	Rectified Linear Unit
RF	Random Forests
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SMOTE	Synthetic Minority Over-Sampling Technique

SSH	Secure Shell
SSL	Secure Sockets Layer
STMP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
Telnet	Teletype Network
TOR	The Onion Router
UDP	User Datagram Protocol
USG	United States Government
USN	U.S. Navy

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Cyber threats against the Department of Defense (DoD) and the greater United States public create an ever-increasing security challenge. Advances in information technology provide new capabilities and benefits but also vulnerabilities. Today, the internet of things (IoT) is almost everywhere. Homes, businesses, hospitals and government organizations are continuing to add internet connected devices for increased productivity and convenience. These devices can be traditional computing devices or more specific sensors for a functional purpose. The DoD will increasingly depend upon a diverse range of IoT devices to gain advantages on the battlefield. Military IoT technology has the promise of helping deliver information dominance over our enemies. IoT technology in real time can provide entity level maintenance, logistics, and intelligence data that has the potential to enable command and control decisions with greater confidence and speed. However, IoT devices are vulnerable to attack (Wrona 2015). Malware has proven to be a continued cause of concern for IoT devices. There have been many high-profile attacks that demonstrate IoT vulnerability. The 2016 Mirai botnet, which amassed a swarm of unsecured IoT devices which delivered a distributed denial of service (DDoS) attack, caused a massive internet outage in the eastern United States (Newman 2017). Other cyber threats such as the SolarWinds breach and the Stuxnet attack highlight the evolving threat and the vulnerabilities of IoT devices.

Advances in machine learning offer potential solutions to detecting the evolving nature of cyber intrusions on internet networks. In this context, our thesis focuses on malware detection, and more specifically, the detection of botnet-infected devices with the use of network flow data in the form of Zeek connection logs. We seek to understand which models are most effective and how they can be deployed for wider use.

We examine a variety of machine learning methods: classification trees, random forests, deep neural networks (DNN), ensemble stacked models and bidirectional long short-term memory (LSTM) neural network models. After deliberate feature engineering we find that each model obtains strong binary classification performance with accuracies ranging from .96 to .998 and Matthew's Correlation Coefficient (MCC) ranging from .86 to .998 . We find that bidirectional LSTM models, trained with an aggregated version of the IoT-23 dataset, achieve the best model performance. Interpreting these results in broader contexts requires

caution but also points to possible opportunities. We find that the IoT-23 network traffic is specific to a homogeneous group of IoT devices. We learn from decision classification trees that simple network rules can be developed for specific IoT devices that have functional purposes. These rules, modified for a specific device, would quickly detect anomalous IoT network behavior triggered by a botnet infection. We suggest that our modeling techniques applied to data collected from networks and devices of interest would benefit efforts to protect DoD IoT devices from the persistent threat of botnet malware infections.

References

Newman L (2017) What we know about friday's massive east coast internet outage. *Wired Magazine*. Accessed, April 15, 2021, https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn.

Wrona K (2015) Securing the internet of things a military perspective. *2015 IEEE 2nd World Forum on Internet of Things*, 502–507 (IEEE).

Acknowledgments

I am incredibly thankful to the U.S. Army for giving me this incredible opportunity to study at the Naval Postgraduate School. There are not many organizations willing to invest so much in an individual solely based upon their commitment to work hard. I look forward to applying the knowledge and skills in ways that benefit our nation, and the U.S. Army

Dr. Robert Koyak, I am incredibly thankful for the countless hours you devoted to helping me complete this thesis. Thank you even more for being a wonderful example of what an Operations Research Analyst should be. I have truly been enriched through our many conservations and your mentorship. LTC Jarrod Shingleton, thank you for your guidance, encouragement, and mentorship through this thesis process.

My family and I will greatly miss First Baptist Church of Monterey. Thank you, Pastor Nate Rehn, Eugene Williams, Cody Cline, Andrey Danilyuk and Bryan Lowry, it was an honor of a lifetime to serve as a deacon at FBCM. Thank you for your continued discipleship, encouragement and teachings that have helped me understand with greater clarity the incredible grace and love of Jesus.

Lastly, to my family. Caitlin thank you for your commitment to our family and unwavering love. You truly put the wind in my sails. Without you all of this would be impossible. Stuart, Sally Jo and Clara, thank you for your understanding for the many hours I needed to dedicate to this program. Stuart, I admire your thirst for knowledge and your love for your sisters. Sally Jo, we love your sharp-witted humor and the love you have for your family. Clara, your smiles light up the world. Mom and Dad, thank you for your love and continued support of my education. Who would have thought I would earn a master's degree in Operations Research!

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Cyber threats against the Department of Defense (DoD) and the greater United States public present an ever-increasing security challenge. Advances in information technology provide new capabilities and benefits but also vulnerabilities. Today, the internet of things (IoT) is almost everywhere. Homes, businesses, hospitals and government organizations are continuing to add internet connected devices for increased productivity and convenience. These devices can be traditional computing devices or sensors designed for a particular purpose. The DoD increasingly depends on a diverse range of IoT devices to gain advantages on the battlefield. Military IoT technology has the promise of helping deliver information dominance over our enemies. In real time, IoT technology can provide entity-level maintenance, logistic, and intelligence data that can facilitate command and control decision making with greater confidence and speed. However, IoT devices also are vulnerable to attack by an adversary (Wrona 2015). Malware has proven to be a continued cause of concern for IoT devices, and there have been many high-profile attacks that demonstrate IoT vulnerability. They include the Mirai botnet, which commandeered a large number of unsecured IoT devices to deliver a distributed denial of service (DDoS) attack that caused a widespread internet outage of the eastern part of the United States (Newman 2017). Other cyber threats such as the SolarWinds breach and the Stuxnet attack highlight the evolving threat and the vulnerabilities of IoT devices.

Advances in machine learning offer solutions to the problem of detecting intrusions of internet networks that evolve with the constantly changing nature of the threats. The purpose of our thesis is to examine the use of machine-learning solutions to detect malware, and more specifically botnet infected devices, with the use of network flow data in the form of Zeek connection logs. We seek to understand which models are most effective and are best suited for wider use.

1.1 Thesis Purpose

The purpose of this study is to contribute to the production of trained machine learning models capable of detecting malware-infected IoT devices using network flow data. We use the Aposemat IoT-23 labeled dataset with malicious and benign IoT network traffic. Our work specifically focuses on detecting malware in the form of botnets. This is considered one of the most pressing IoT security concerns to date. Our methods of developing malware classification algorithms may help guide machine learning solutions for specific network applications. We not only seek to develop models but also analyze why they work and if our methods are general enough for wider use.

1.2 Research Questions

This study examines the behaviors of IoT devices infected with malware botnets. Our ultimate goal is the development of robust and generalizeable machine learning models able to quickly determine if an IoT devices has been compromised to botnet infection. By analyzing and developing various supervised machine learning models we seek to answer the following questions:

1. Which machine learning algorithms are well suited for malware detection using network flow data?
2. Which data features are best suited for machine learning malware botnet detection?
3. Are machine learning algorithms trained with the Iot-23 data set generalizable for wider use?

1.3 Scope, Limitations and Assumptions

Our primary interest is the detection of IoT devices that have been infected specifically with malware botnets, solely with the use of network flow data in the form of Zeek connection logs.

Our work primarily relies upon the open source IoT-23 dataset. This dataset captures a wide variety of IoT infected botnet traffic, but it is limited to only a few different IoT devices on one particular network. This work does not encompass all forms of malware; rather it

focuses specifically upon botnets.

We assume the IoT-23 dataset represents the general network behaviors of IoT devices infected with malware botnets. We accept that intricacies of specific IoT devices, hosted on certain networks may differ from IoT devices recorded within the IoT-23 dataset. General concepts and methods developed by our work can be applied to specific use cases.

1.4 Thesis Organization

The remaining parts of this thesis are structured as follows: Chapter II provides background information relevant to this study by briefly describing IoT, IoT vulnerabilities, how network data is collected, an analysis of malware with specific information concerning botnets, and past works concerning machine learning and malware detection. Chapter III describes the IoT-23 dataset and methodology used in this thesis. Chapter IV presents our conclusions from decision tree (DT), random forests (RF), deep neural networks (DNN), and long short-term memory (LSTM) neural network machine learning models. Chapter V describes our conclusions, recommendations, and suggestions for future research concerning botnet malware classification.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

2.1 Military Internet of Things "MIoT"

The internet of things can be broadly described as a network of internet connected devices. The term is often used to describe sensors and devices that communicate with other devices and outside servers. IoT has become ubiquitous with domestic households as TVs, grills, smart speakers, door locks, security systems, smartphones, watches and cameras connect to internet networks. IoT is not just a phenomenon in American households it has widespread applications in industry and government. For example the Lower Colorado River Authority have deployed sensors along the Colorado river to help track increasing stream levels that would indicate the possibility of flooding (Harbert 2017). The tech company Zubie provides vehicle fleet monitoring through IoT devices that enable preventive maintenance tracking, allowing for enhanced resource management. The military application of IoT (MIoT) as is true in civilian domains, will become increasingly prominent as network-centric warfare and information dominance grow in greater importance. Some applications of MIoT are the use of IoT devices in logistics and maintenance, soldier healthcare sensors, C4ISR to enable battlefield awareness, fire-control systems to enable automated response to threats and energy management on military installations (Zheng and Carter 2015). The promises of MIoT are coupled with the apparent vulnerabilities of MIoT (Wrona 2015). These vulnerabilities are discussed further below. It is important to recognize that MIoT presents both opportunities and risks.

2.2 Network Data Collection

A computer network comprises of two or more computers that are connected by a physical layer (cables, or WiFi) for the purpose of transmitting, exchanging, or sharing data and resources. Networked devices communicate upon protocols known as the Transmission Control Protocol / Internet Protocol (TCP/IP) suite. With these protocols packets of information are sent across networks and reassembled by the receiving device. The network communications process can produce a tremendous amount of data. There are two overarch-

ing ways to capture network data, Packet Capture (PCAP) and network flow capture. PCAP records the actual packets and can be used to view packet content and even reconstruct previous internet traffic (Horneman and Dell 2014). This collection method generates large volumes of data. For medium sized networks, PCAP data may be collected in terms of gigabytes per second. Over time, PCAP data becomes burdensome to store and analyze. Another method of analyzing network traffic is through the collection of network flow data. A flow is a set of IP packets between a source port and destination port that provides series of summary statistics describing the data transaction. The summary statistics often include the number of packets, the total number of bytes and duration. Collected network flow data is much more efficient for long term storage and analysis. The nature of network flow data provides manageable data sources that can be used for machine learning algorithms. There are several popular software applications that capture network flow data, including Cisco's Netflow and Zeek.

2.2.1 Zeek/Bro

Zeek, formally known as Bro, is an UNIX based open-source software network analysis framework. Zeek was developed for security monitoring and traffic analysis by Vern Paxson at the University of California, Berkeley (Paxson 1999). Zeek produces data efficient logs that capture network activity. It is also a Turing complete programming language which enables the analyst to query and customize data collection. Zeek logs reference each observation as a network connection. A connection is defined as a network flow between a so called "socket pair." Each "socket pair" consists of a host IP address and port and a destination IP address and port. Each Zeek network flow connection provides summary statistics such as the number of bytes sent or received, the duration of the connection and the protocol.

Zeek produces 60 different log files capturing 400 different network variables. The log files capture key information pertaining that include but not limited to HTTP, DNS key headers and SSL. Zeek's cornerstone log is the connection log. This log tracks key statistics for both TCP and UDP connections to answer the basic questions of "who is talking to whom, when, for how long and with what protocol" (Zeek 2021).

The use of Zeek has become widespread in the network security community. The U.S. Army

NETCOM, as well as many other organizations extensively, use Zeek to monitor network behavior.

Table 2.1. Zeek Connection Log. Source: Dutta et al. (2020) and Zeek.org.

	Features	Description
1	fields-ts	Flow start time
2	uid	ID Code
3	id.orig-h	Source IP address
4	id.orig-p	Source port
5	id.resp-h	Destination IP address
6	id.resp-p	Destination port
7	proto	Transaction protocol
8	service	http, ftp, smtp, ssh, dns, etc.
9	duration	Duration of connection
10	orig-bytes	Source to destination transaction bytes
11	resp-bytes	Destination to source transaction bytes
12	conn-state	Connection state
13	local-orig	Source local address
14	local-resp	Destination local address
15	missed-bytes	Missing bytes during transaction
16	history	History of source packets
17	orig-ip-bytes	Flow of source bytes
18	resp-pkts	Destination packets
19	resp-ip-bytes	Flow of destination bytes
20	tunnel-parents	Traffic tunnel

2.3 Malware

Malware is an overarching term for harmful or intrusive software intentionally designed to damage or destroy computer systems as well as compromise data integrity. It is used for a variety of nefarious actions to include DoS & DDoS attacks, targeted intrusions, and ransoming of information systems. Malware comes in many forms; different malware

classifications are based upon the means of delivery, and the intended purpose. Some of the most common classifications of malware are: adware, spyware, viruses, worms, trojans, keyloggers, rootkits, botnets, and ransomware. Some of the classifications are exclusive while others have overlapping qualities. Viruses infect other files and computers but require the user to run an executable file. Worms are similar to viruses but they do not require user interaction. Trojans, unlike viruses and worms, do not seek to propagate. Their intended purpose is to open system access for a hacker.

IoT devices are susceptible to malware infections and are capable of being employed to conduct cyber-attacks. Some IoT devices are considered appealing targets because they do not always support strong security mechanisms and are unlikely to patch software vulnerabilities. Vulnerable IoT devices can act as an entry points into a network or be the purpose of the attack (Meneghello et al. 2019). One of the most challenging and consequential security issues for IoT devices are botnet infections.

2.3.1 Botnets

Botnets refer to a network of computers or IoT devices that are infected by malware and are under the control of an attacking agent or "bot master". Infected devices then seek to find vulnerabilities in other devices in order to further propagate the span of the botnet. Once the bot master has a sufficient number of infected bots it is capable of launching cyber-attacks (Wang et al. 2008). Botnets can be considered a combination of both a worm and a trojan.

The Mirai botnet first appeared in 2016 and has been used in several large-scale DDoS attacks after its creator released the source code to the public (Herzbeg et al. 2016). A notable Mirai botnet attack was against the DNS provider Dyn. The Mirai botnet was used to amass an estimated 140,000 IoT devices for the purpose of a DDoS attack (Bursztein 2017). During the attack Dyn's servers were receiving requests at 1.2 terabytes per second from the army of bots (Woolf 2016). The Mirai botnet disabled hundreds of well-known websites including Amazon, Paypal, Twitter, Netflix, Reddit and Github for several hours (Woolf 2016). The intended target was believed to be video game servers of the popular game Minecraft, the well-known websites were likely unintended collateral damage of the attack (Bursztein 2017).

Months later a variation of the Mirai botnet was used by other actors to attack the Liberian

telecommunications company Lonestar Cellular (Chellel 2019). Chellel (2019) notes that the Lonestar attack primarily used infected WiFi enabled web cameras to launch DDoS attacks. The attack resulted in thousands of customers canceling services with Lonestar Cellular. During the Lonestar cyber-attack, the attempt to infect new devices created massive internet outages when 900,000 routers owned by the German Internet provider Deutsche Telekom were infected (Musil 2016). When the routers were infected, they crashed due to unintended bugs in the malware binaries (Chellel 2019; Musil 2016).

Mirai botnets as well as other botnet variants establish command and control (C&C) servers that provide the botmaster with a centralized management tool to coordinate new attacks. The C&C server establishes periodic communications with infected devices to log availability statuses and provide instructions for attacks (Kolias et al. 2017). Discovered C&C servers communicate to bots using the IRC, TCP, HTTP and POP3 protocols (Ryu et al. 2018). C&C servers conceal their identity through the use of the Onion Router (TOR) and proxy servers. Botnets establish centralized (depicted in Figure 2.1) or peer-to-peer communication networks depicted in Figure 2.2. Centralized bot networks operate with each device communicating directly back to one of the botmaster's C&C servers. Peer-to-peer bot networks differ in that the bot master's communications are distributed through the network of bots. Peer-to-peer communication is a more sophisticated method that conceals the C&C servers. In order for the infected bot to block commands of the C&C servers it would need to block connections to every other bot in the network (Wang et al. 2010; Grégio et al. 2015).

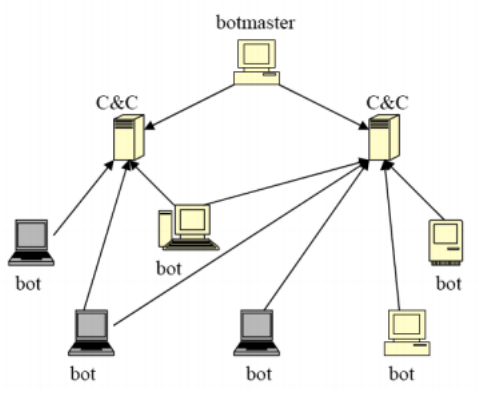

Figure 2.1. Centralized Bot Network. Source: (Ryu et al. 2018).

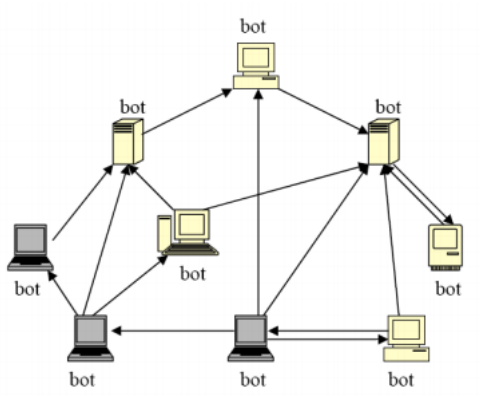

Figure 2.2. Peer-to-peer Bot Network. Source: (Ryu et al. 2018).

After establishing the C&C servers the botnets then scan public IP addresses looking for vulnerabilities and access into local networks. When vulnerable IoT devices are found, the botmaster through automation issues commands and payloads necessary to compromise the integrity of the device. After a newly infected device is under control of the botnet, it often closes SSH and Telnet ports as measures to protect itself from other malwares (Kolias et al. 2017). Once the device is infected it remains in one of four states which are depicted in Figure 2.3. In a dormant or benign state, the bot communicates with the C&C server waiting for commands. On command the botmaster can issue propagation commands that instruct the device to find new devices. First the bot must discover other devices to infect through of port scanning. Once a vulnerable device is found the bot begins to attack the device in order to further the span network of the botnet. The original Mirai botnet relied primarily upon the Telenet protocol TCP ports 23 and 2323, but new variants have become more sophisticated using other ports and protocols making detection more challenging (Kolias et al. 2017). In attack mode the bot becomes part of a DDoS attack against intended targets. During this phase the botnet infected devices send many requests to a server in an attempt to overload and disable it. Botnets often generate numerous network flow connections during the propagation port scanning, and attack DDoS phases.

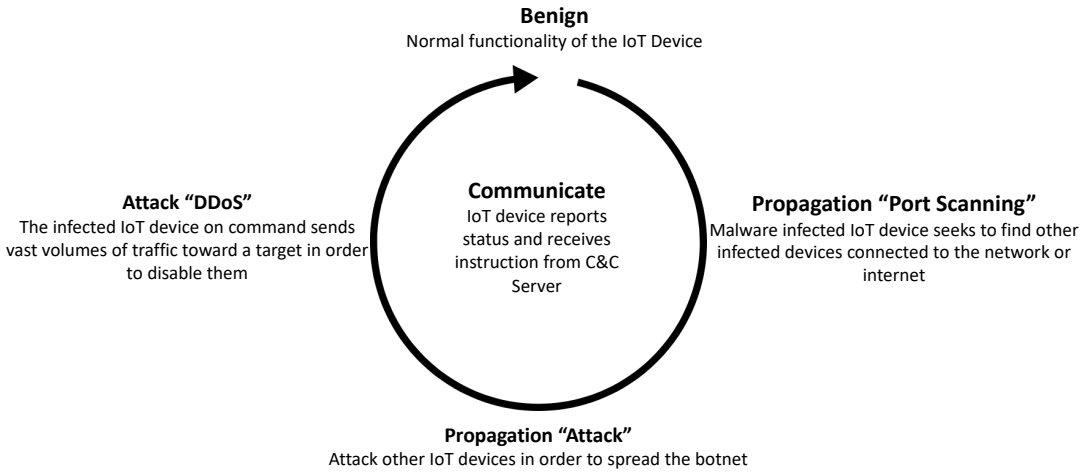

Figure 2.3. Botnet Activity Cycle

Botnets present security concerns that are apparent to DoD. Current and future MIoT have the potential of becoming infected and disabling key networks. MIoT sensors purchased by the DoD may have zero-day vulnerabilities and have the potential of being infected with malware that would behave in similar ways to the Mirai botnets and its varying manifestations. At strategic moments DoD networks could experience denial of its services which may have catastrophic outcomes. The DoD also has an interest in protecting the United States in the cyber domain. The examples highlighted thus far have been carried out by low level and at times clumsy criminals. Nation states and non-state actors have the potential to use botnets with greater sophistication that could target attacks against key commercial and civic organizations. Large-scale use of botnets could have profound effects on American commerce and governance at key moments. Due to the threats posed by botnets, developing new means for detection is of great relevance and importance to the DoD.

2.4 Machine Learning and Malware Detection

Past work related to the use of machine learning to detect malware can be categorized by two approaches: static and dynamic (Gibert et al. 2020). Static techniques seek to analyze the code structure and other data properties of the malicious binary. Dynamic techniques seek to analyze the behaviors of executed malware on a device or network.

There have been several interesting studies that demonstrate the use of static techniques. Naeem et al. (2018) propose a malware image classification system that is designed to transpose malware binaries onto a gray-scale image. The gray-scale images are then used to train supervised machine learning algorithms for malware detection. The authors obtain 97.4% accuracy detecting 25 different malware variants. Other static techniques have achieved similar rates of accuracy. However, there are disadvantages to this approach. The malware must be known and analyzed prior to detection so it can be incorporated into the machine learning training dataset. This method is also prone to failure due to advances in code obfuscation techniques and file-less malware which is only represented as an abstraction in the system's memory (Saad et al. 2019; Gibert et al. 2020).

Dynamic approaches can be further divided into two categories. The first is instruction traces. This method seeks to analyze how file structures change, and how malware uses the infected devices Central processing unit (CPU) and memory. The other dynamic method which is the focus of this thesis is a network-based approach. Network based approaches seek to analyze the network traffic behaviors of infected devices. In past work network-based approaches have often focused on the use of network flows data to detect traffic of malware infected devices.

Ryu et al. (2018), explores the use of a suite of supervised machine learning algorithms to detect botnet infected devices with network flow data. The study uses the CTU-13 labeled NetFlows dataset for training. Ryu et al. (2018) discusses the challenges of imbalanced response classes often present in labeled malware detection datasets and proposes Matthew's Correlation Metric (MCC) as a primary performance criterion. The study finds that the Random Forests algorithm and decision tree classification to consistently produce the best performing models both in terms of MCC and training time.

Dutta et al. (2020), present a binary classification ensemble model that incorporates a

Deep Sparse AutoEncoder (DSAE), a deep neural network (DNN), and a long short-term memory (LSTM) model. The authors use the DSAE for dimensionality reduction, then use predictions from both the DNN and LSTM as inputs to a logistic regression meta-classifier. Their model is trained and evaluated with three labeled network flows datasets, IoT-23, LITNET-2020 and NetML-2020. Dutta et al. (2020) only use a small subset of the IoT-23 dataset due to the volume of the data. The model achieves high performance metrics. The article does not explore the generalizability of their model to new networks and devices. This article was published during the development of this thesis and is interesting because it is one of the first publications using the IoT-23 dataset.

There are only a few published works that specifically address the use of Zeek logs for machine learning. Gustavsson (2019), demonstrates the use of several machine learning classifiers with customized features created through the Zeek programming language. His work demonstrates the flexibility of Zeek to produce streaming robust machine learning features available in real time. Like other studies Gustavsson (2019) finds Random Forests to obtain better performance than other machine learning algorithms.

Meidan et al. (2018) examine unsupervised techniques for malware detection. Their journal article focuses specifically on IoT network traffic subject to infection from the Mirai and Gafgyt botnets, and the use of Deep Autoencoders. This study is distinguished by its use of autoencoders as fully automated standalone malware detectors. The most common use of autoencoders is for dimensionality reduction, outlier detection, and labeling for semi-supervised machine learning. Their model seeks to capture normal IoT device traffic for specific IoT devices as baselines of normal activity within a time window. When traffic deviates from normal the model indicates possible malicious activity. The study incorporates the use of various home IoT devices infected by brute force Telnet attacks. The autoencoder model in most cases successfully detects botnet traffic within seconds of infection, with reportedly few false positives. The authors find that IoT devices with more complicated network traffic patterns lead to higher false positive rates. The study emphasizes that IoT devices with predictable network traffic patterns can be profiled for normal and malicious network behavior.

The use of machine learning in cyber security has been widely studied in technical literature. Google scholar indicates as many as 47,200 publications concerning machine learning and

malware detection. The works we describe represent topics most informative to our work concerning machine learning, network flows, the use of Zeek and botnets. Our review of the literature indicates that tree-based machine learning algorithms and LSTM models offer promising results. Lacking in any of the literature known to us are inferential perspectives concerning model performance. Most studies simply present algorithms but fail to explain why they work or fail to work. Knowing the reason for model performance is crucial to understanding the model's generalizability to a wider array of network activity.

2.5 Malware Open Source Datasets

There are several labeled open-source malware datasets. The IoT-23 dataset is the only labeled dataset built upon the framework of Zeek connection logs. The IoT-23 dataset will be described more in detail in the following chapter. Other network flow datasets such as DARPA 98, LITNET-2020, NetML-2020, KDD Cup 1999 are offered in Cisco NetFlow's logs, PCAP files, or in customized network flow extractions from PCAP data. Labeled datasets are useful because they allow the analyst to perform a wide range of analysis to include supervised machine learning. Supervised machine learning requires data labeling as a means to measure loss between predicted values and true values. The loss function is then optimized to minimize prediction error. Labels in malware dataset come in varying forms. Some labels classify particular network traffic as either benign or malicious, other labels include particular malicious behaviors exhibited in the network traffic and others specify the variant of malware. Labeling of the datasets is conducted by subject matter experts (SME) during the recording of network traffic. The SME due to the large nature of the data develops scripts based upon analysis of the malware to facilitate the labeling process (Hussain et al. 2020). Another way analyst label datasets is by simply recording the time of the attacks during controlled experiments. This method is common when data consist solely of PCAP files which are ill-structured for direct labeling.

Though network captures of malware are well suited for machine learning model development there are drawbacks. Labeled datasets can only record known or created malware traffic. Since there is a wide array of malware variants it is impossible to fully capture all known or future malware variants. Another limitation is that malware datasets are influenced by the devices used during the controlled experiment (Hussain et al. 2020). There is a wide range of IoT devices currently in use, each of whom use specific applications and have

certain functionalities. Some of these limitations can be mitigated by attempting to classify specific malicious behavior rather than specific malware variants and to retrain models for specific IoT devices used in a production environment.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Data and Methodology

This chapter provides an overview and discussion of the IoT-23 dataset, important aspects of data cleaning, and feature engineering. It also describes methods used in our research for mediating the effects of class imbalances in the data and discusses appropriate performance metrics for the evaluation of machine learning models. Finally, this chapter provides an overview of the machine learning algorithms (random forests and deep neural networks (DNN)) used for malware detection on the IoT-23 dataset.

3.1 Analysis of IoT-23 Dataset

The primary data used for our research is the open source Aposemat IoT-23 (Parmisano et al. 2020) labeled dataset. The IoT-23 dataset contains 20 different malicious malware captures and three benign IoT device captures (see Tables 3.2 and 3.3). Stratosphere Laboratory generated the dataset at the Czech Technical University (CTU) and made the data available to the public in January 2020. Stratoshpere Laboratory obtained each of the 20 data captures by infecting an IoT device with a real-world malware variant. The entire IoT-23 dataset consists of approximately 300 million network flows. We chose this for its accessibility, relevancy and labeling. The data being open source allows for easy access on both personal and NPS computing resources. The data set is relevant because its malware variants are seen on actual IoT devices. The most important aspect of the data is its labeling (i.e., known disposition of records as either infected with a particular type of malware or benign) that provides an opportunity for use of supervised machine learning algorithms. Supervised machine learning algorithms use labeling or known truths to optimize models for predicting the disposition of newly introduced, unlabeled data.

Parmisano et al. (2020) collected the IoT-23 dataset with raw PCAP and Zeek connection logs (see chapter 2 section Zeek/Bro). The Stratosphere Laboratory(2020) implemented an automated labeling scheme recorded as additional fields of the Zeek connection logs. The dataset provides two labeled response vectors. The first is simply a binary label [malicious, benign]. The second is a more detailed label consisting of the following label categories (Parmisano et al. 2020):

- **Attack:** indicates an attack originating from a botnet infected device to another device. The attacking IoT device is seeking to exploit known network vulnerabilities. Attack methods include brute force attack on a telnet login, and command injection.
- **Benign:** indicates no malicious network flow connections.
- **C&C:** indicates a network flow from an infected device connected to a command and control (C&C) server.
- **DDoS:** indicates that a network flow from an infected device was executing a Distributed Denial of Service (DDoS) attack against another device.
- **FileDownload:** indicates an infected file was download.
- **HeartBeat-C&C:** indicates that packets sent during the network flow connection were being used to track an infected device by a HeartBeat C&C server.
- **Mirai-C&C:** indicates that the network flow connection has characteristics of a Mirai botnet C&C server.
- **Okiru:** indicates that the network flow connection has characteristics of an Okiru botnet.
- **PartOfAHorizontalPortScan:** indicates that a network flow connection is part of a horizontal port scan. Port scans are used to identify vulnerabilities in targeted devices.
- **Torii-C&C:** indicates that the network flow connection has characteristics of a Torii botnet C&C server.

The detailed labels contain two subcategories of labels. The first category consists of behaviors of a malware infected device. These behaviors are downloading infected binary files, communicating with a C&C server, port scanning to find vulnerabilities in other IoT devices, attacking or conducting DDoS in order to suppress a targeted device or server. The other subcategory contains labels that are associated with the Okiru botnet. Okriu is a Mirai botnet variant and it's unclear what the label means in terms of actions taken by the infected device. For this reason, malware captures 43-1, 17-1, 36-1, 33-1, and 7-1 which contain

the Okriu label present a challenge for multi-class classification problems. One solution is simply to remove these captures or filter Okriu connections from the dataset.

The dataset classes are unbalanced, posing a challenge to machine learning classifiers. Malicious traffic composes the majority of observations with benign traffic consisting of only 10% of the data. The primary reason for the class imbalance in the dataset is that DDoS, port scanning and connections associated with the Okiru botnet generate large volumes of network flow connections in small time increments. Conversely, the FileDownload label is extremely sparse with only a total of 53 flows out of the 300 million observations in the entire dataset. This imbalance can create challenges for machine learning classifiers as we discuss in section 3.4. For additional reference Table 3.2 provides the number of detailed label occurrences respectively.

Table 3.1. Detailed Label Counts

	Label Detail	Counts
1	PartOfAHorizontalPortScan	213,852,924
2	Okiru	47,381,241
3	Benign	30,858,735
4	DDoS	19,538,713
5	Okiru-Attack	13,609,470
6	C&C-HeartBeat	33,673
7	C&C	21,995
8	Attack	9,398
9	C&C-PartOfAHorizontalPortScan	888
10	C&C-HeartBeat-Attack	834
11	C&C-FileDownload	53
12	C&C-Torii	30
13	FileDownload	18
14	C&C-HeartBeat-FileDownload	11
15	PartOfAHorizontalPortScan-Attack	5
16	C&C-Mirai	2

A total of four unique IoT devices were used in the formation of the dataset. In the malware captures a Raspberry-Pi was used as the infected device. A Raspberry-Pi is a small modifiable computing device popular among hobbyist. The benign honeypot captures used the following IoT devices for data collection: Amazon Echo, Philips HUE Smart Light Bulb and the Somfy Door Lock. The Raspberry Pi device offers advantages and disadvantages in the creation of training datasets for malware detection. One advantage is that a Raspberry Pi can operate on the Linux operating system. Most IoT devices use one of the many Linux distributions available from the open-source community. The problem with using a Raspberry Pi as the infected device is that it is designed for access by the end user. Easy end-user access has the potential to create vulnerabilities that may be too trivial and unrealistic compared to a real-world scenario, which could impart bias to the dataset. For instance, the Raspberry Pi could leave an open and unsecured Telent or SSH port known to the dataset

creator. Secondly, the problem with only using a Raspberry Pi is that it may not represent the computing capacity or normal behaviors of IoT devices in general. The network flow connections represented in a Zeek connection log may produce more variability than a small lightweight IoT device such as a smart light bulb. These points must be considered when constructing a machine learning classifier for generalizations used in a production environment.

The IoT 23 dataset consists of several variations of botnets infections. The Mirai botnet is the most represented (Table 3.2). IRCBot, Kenjiro, Gagfyt, and Muhstik botnets demonstrate similar behaviors as the Mirai but have different targets or modify communication protocols. The Mushstik botnets propagate through routers using the open source firmware Tomato (Hilt et al. 2020). The Hajime botnet is similar to the Mirai but more sophisticated using a peer-to-peer C&C server method. The botnet's purpose is unknown and somewhat mysterious. When it infects a device it seeks to remove other botnets, blocks ports 23, 5358, 5555, 7547, and then continues to propagate. Hajime leaves in a log file the message "Just a white hat, securing some systems"(Herwig et al. 2019). The Torii and Hide and Seek botnets are unique because they are able to survive system reboots. The Torrii botnet's intended purpose appears not to attack targets with DDoS rather it seeks to exfiltrate sensitive information using "wget", "ftpget", "ftp", or "busybox wget" (Akub Kroustek and Hron 2018).

Table 3.2. IoT-23 Datasets with Malicious Activity

Name of Dataset	Duration(hrs)	#ZeekFlows	Malware	Ratio Benign
Malware-Capture-39-1	7	73568982	IRCBot	1e-04
Malware-Capture-43-1	1	67321810	Mirai	0.30562
Malware-Capture-17-1	24	54659864	Kenjiro	0.00058
Malware-Capture-33-1	24	54454592	Kenjiro	0.02536
Malware-Capture-52-1	24	19781379	Mirai	9e-05
Malware-Capture-36-1	24	13645107	Okiru	2e-04
Malware-Capture-7-1	24	11454723	Linux.Mirai	0.00663
Malware-Capture-35-1	24	10447796	Mirai	0.79083
Malware-Capture-9-1	24	6378294	Linux.Hajime	0.00354
Malware-Capture-49-1	8	5410562	Mirai	0.00068
Malware-Capture-60-1	24	3581029	Gagfyt	0.00069
Malware-Capture-48-1	24	3394347	Mirai	0.0011
Malware-Capture-1-1	112	1008749	Hide and Seek	0.46521
Malware-Capture-3-1	36	156104	Muhstik	0.02906
Malware-Capture-34-1	24	23146	Mirai	0.08308
Malware-Capture-8-1	24	10404	Hakai	0.20965
Malware-Capture-42-1	8	4427	Trojan	0.99864
Malware-Capture-21-1	24	3287	Torii	0.99574
Malware-Capture-20-1	24	3210	Torii	0.99501
Malware-Capture-44-1	2	238	Mirai	0.8903

Table 3.3. IoT-23 Datasets without Malicious Activity

Name of Dataset	Duration(hrs)	#ZeekFlows	Device
Honeypot-Capture-7-1	1.4	139	Somfy Door Lock
Honeypot-Capture-4-1	24	461	Philips HUE
Honeypot-Capture-5-1	5.4	1383	Amazon Echo

3.2 Data Cleaning

We conducted data cleaning activity primarily in the R programming language (R Core Team 2020) using the NPS Hamming HPC. First, the Zeek connection logs were parsed into R data frames. Then the features local-orig, local-resp, and tunnel-parents were removed due the vast majority of their observations being incomplete. Observations where the duration, bytes or packets were incomplete were replaced with the value zero. After the initial data cleaning was completed the data was saved to a series of comma separated values (CSV) files.

In order to use Python programming language machine learning software (API Scikit-learn and TesnsorFlow Keras API), categorical variables must be represented as numeric values. We converted the categorical variables to numeric values using one-hot-encoding. Response variables were converted to numeric values by ordinal numeric encoding.

3.2.1 Scaling and normalizing data

Neural network software usually benefits from scaling and normalization of numeric input features to ensure faster convergence and better solutions (Stottner 2019; Hale 2019). Four widely-used methods available in the Scikit-Learn API are min-max scaler (3.1), robust scaler (3.2), standard scaler (3.3) and normalizer (3.4). To describe these methods we let x, y and z represent numerical feature vectors and x_i, y_i, z_i represents individual values within the vectors.

The min-max scaler depicted in equation (3.1) is considered a standard starting point for DNN and most machine learning applications. It scales the values between a specified range with the default being [0,1]. This function preserves the shape of the original distribution and maintains the importance of outlier values.

$$MinMaxScaler(x_i, x) = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.1}$$

The robust scaler incorporates inter-quartile range. This scaler reduces the influence of

outliers within a vector.

$$RobustScaler(x_i, x) = \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{3.2}$$

The standard scaler is used to transform the feature vector to an approximation of the normal distribution with mean zero and unit variance.

$$StandardScaler(x_i, x) = \frac{x_i - mean(x)}{stdev(x)} \tag{3.3}$$

Normalizing works differently because it transforms the data by rows instead of columns.

$$Normalizer(x_i, y_i, z_i....) = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2 + ...}} \tag{3.4}$$

3.3 Feature Engineering

After initial data cleaning, we derived several new features from the data. The purpose of feature engineering is to create features generalizable to new data, to limit the dimensionality of categorical variables, to incorporate domain knowledge, and to account for time dependency within and between network flow connections. Table 3.4 provides a summary of our proposed engineered features.

Table 3.4. Engineered Features

	Engineered Features	Description
1	time_diff_from_last_connection	Time difference from last connection
2	bytes_per_sec	Total bytes per second
3	bytes_per_sec_orig	Source bytes per second
4	bytes_per_sec_resp	Destination bytes per second
5	pkts_per_sec	Total packets per second
6	hist_str_len	Number of characters in history field
7	orig_class	Source IP address class
8	resp_class	Destination IP address class
9	orig_port_level	Source port class
10	resp_port_levels	Destination port class
11	ShAd_hist	History String pattern ShAd
12	ShAD_hist	History String pattern ShAD
13	ShAF_hist	History String pattern ShAF
14	num_flows	Number of client flows last 10th second
15	orig_bytes_total	Total source bytes last 10th second

Before modeling it is important to ensure that the dataset is generalizable, such that a specific IP address and port combination would not bias machine learning algorithms. We bin the IP addresses contained in the features id.org-h, and id.resp-h into the following internet protocol version 4 (IPv4) address classes: A,B,C,D,E and other. Class "other" contains internet protocol version 6 (IPv6) addresses. The IPv4 address classes (A through C) are based upon the network size hosting the IP address. Class A is reserved for very large networks that offer a large number of possible IP addresses. Classes B and C are used for smaller networks. Class D address are not assigned to host but used for mulitcasting and class E is reserved for research purposes. Table 3.5 represents the the IP address ranges for each class.

Table 3.5. IP Address Classes

	Class	Address Range
1	Class A	1.0.0.1 to 126.255.255.254
2	Class B	128.1.0.1 to 191.255.255.254
3	Class C	192.0.1.1 to 223.255.254.254
4	Class D	224.0.0.0 to 239.255.255.255
5	Class E	240.0.0.0 to 254.255.255.254
6	Other	IPv6 Address

We bin port numbers into three categories: well known ports, registered port and dynamic ports. Well known ports are reserved for specific TCP/IP applications such as HTTP, HTTPS, and SSH. Registered port numbers are ports used by organizations for unique persistent server applications. Often these ports are registered with Internet Assigned Numbers Authority (IANA). One example of a registered port is UDP 1716, which is designated for online video game America's Army developed for the U.S. Army recruiting command. Dynamic ports are not controlled and intended for temporary use. Dynamic port numbers are used typically by clients and not servers (Goralski 2017). Port number ranges are depicted in Table 3.6.

Table 3.6. Port Categories

	Category	Port Number Range
1	Well Known Ports	0-1023
2	Registered Ports	1024-49151
3	Dynamic Ports	49152-65535

The Zeek connection logs contains a history field that provides a description of events that occurred in the TCP or UPD connection. This field, which consists of a series of characters that are represented in a ordered string, captures the process known as the "three way TCP handshake" or a corruption of that process. We analyze labels of malicious traffic

for common patterns presented in the history field. Flows that contain history field values ShAd, ShAD and ShAF appear to be associated with malicious labels. Based on the analysis of the history field, we create three new binary fields that contained these patterns. These labels can be decoded to give actual network meaning. Capital letters refer to actions taken by the source client IP and lower case refers to the destination client IP. For example, ShAd translates to the following sequence of events that occurred during the connection:

1. S- The source IP sent a SYN segment,
2. h -The destination IP sent a SYN ACK segment
3. A- The source IP sent an ACK segment.
4. D- The source IP sent at least one segment with payload data.

We derive several variables to associate values with time dependency. Variable 1 (Table 3.4) measures the time difference between the current flow and the last. Variables 2-4 captures rates of specified features in units per second. Time lag features num_flows and orig_bytes_total capture time dependencies in a series of connections. This is useful for identifying connections associated with malicious events that generate many observations such as DDoS and Horizontal Port Scanning. The two time lag features count the number of connections and sum the number of source bytes associated with the current connection IP address in the the past 10th of a second. The algorithm used to produce this feature was implemented in Python (see NPS GitLab page).

We developed all of the features in this section in the R (R Core Team 2020) or Python (Van Rossum and Drake Jr 1995) programming languages. For use in production these features could be replicated in the Zeek programming language to efficiently modify Zeek connection logs.

3.4 Data Label Imbalances

The IoT-23 dataset has imbalanced response variables. The imbalance stems from an unequal amount of benign and malicious labeled network flow connections. Data imbalances are a common phenomenon in machine learning classification problems and can be problematic. Imbalanced datasets are often problematic for supervised machine learning because the algorithm is not equally penalized for classifying minority classes (Chawla et al. 2004).

Therefore, data imbalances often create a classification bias in favor of the majority class. The problem is further exacerbated when a trained supervised machine learning model is used to make predictions with new data having a different distribution of response variables. Network traffic prior to botnet infection has few if any malicious flows. After infection infected flows become the majority of flows. There are several methods that can be taken in order to mitigate the negative effects sometimes attributed to machine learning classifiers trained with imbalanced data sets. These methods are up sampling the minority or down sampling majority response variables, synthetic creation of data, or the use of class weights.

- **Down Sampling or Up Sampling:** Down sampling is the method where a subset of the majority class is sampled without replacement. This can be an effective method, but often valuable data is omitted from the machine learning process. Up sampling refers to randomly sampling from the training dataset with replacement. This can be problematic because observations are then duplicated. Duplicated observations if replicated too often can create an artificial bias (Chawla et al. 2004). Another disadvantage of up sampling with large data sets is explosive growth of the size of the training data.
- **Synthetic Minority Over-sampling Technique (SMOTE)** SMOTE creates additional observations of the minority class through the use of the K-nearest neighbors algorithm (Chawla et al. 2002). This is a method that has been effective in many machine learning case studies. The API Scikit-Learn includes commonly known SMOTE algorithms.
- **Class Weights:** Another method to mitigate the effect of class imbalance is to provide greater weight to the minority response class. Weights allow the loss function to penalize with greater magnitude miss-classifications of minority labels. The naive approach to assigning class weights is simply by using the proportions of the classes found in the training dataset. This sometimes fails to increase model performance. Different weighting ratios must be experimented with in order to find improved results. Class weights can be problematic especially if the class proportions in a real-world application may differ(Huang et al. 2013). In the case of IoT malware detection, if the device is not infected benign observations will be in the majority. However, when IoT devices are infected, the opposite would be true.

3.5 Data Sampling Methods

We partitioned the IoT-23 dataset into testing and training based on a random 40/60 split. A random split was used in order to capture observations from all of the 23 different IoT malicious and benign captures. The trainset partition, due to the size of the data, allowed for the creation further validation splits.

Due to the large size of the IoT-23 dataset, random subset sampling of training data was used in order to experiment with different algorithmic parameters in a time efficient manner. It was generally found that a random subset consisting of only 10% to 15% of the entire dataset produced results similar to using the entire training data partition in both random forest and neural network models.

3.6 Data Aggregation Methods

Data aggregation was also considered in order to reduce the number of observations. Aggregation in some IoT-23 captures helped balance the data by reducing network flows the IoT devices generated in quick succession. We collected summary statistics by aggregating time windows of a 10^{th} of a second for a particular IoT Device. The response variable was considered malicious if at least one observation in the time window was classified as malicious. This aggregation method reduced the number of observations from 300 million to 2.3 million observations. The total file is reduced to 2.4 GB from the original 50 GBs. A summary of the aggregated data is provided in Table 3.7.

Table 3.7. IoT-23 Dataset Aggregated by 10^{th} of Second Time Windows

	Name of Dataset	#Observations	Malware/Device	Ratio Benign
1	Malware-Capture-1-1	440734	Hide and Seek	0.21
2	Malware-Capture-17-1	344255	Kenjiro	0.05
3	Malware-Capture-35-1	334874	Mirai	0.99
4	Malware-Capture-33-1	330918	Kenjiro	0.02
5	Malware-Capture-48-1	139444	Mirai	0.01
6	Malware-Capture-39-1	130178	IRCBot	0.01
7	Malware-Capture-7-1	111545	Linux.Mirai	0.08
8	Malware-Capture-36-1	110487	Okiru	0.02
9	Malware-Capture-3-1	97070	Muhstik	0.02
10	Malware-Capture-9-1	96350	Linux.Hajime	0.04
11	Malware-Capture-49-1	72095	Mirai	0.01
12	Malware-Capture-52-1	67288	Mirai	0.02
13	Malware-Capture-43-1	36799	Mirai	0.07
14	Malware-Capture-8-1	10365	Hakai	0.21
15	Malware-Capture-34-1	8776	Mirai	0.21
16	Malware-Capture-60-1	4000	Gagfyt	0.12
17	Malware-Capture-20-1	2935	Torii	0.99
18	Malware-Capture-42-1	2911	Trojan	0.99
19	Malware-Capture-21-1	2579	Torii	0.99
20	Honeypot-Capture-5-1	804	Amazon Echo	1.00
21	Honeypot-Capture-4-1	350	Philips Hue	1.00
22	Malware-Capture-44-1	235	Mirai	0.89
23	Honeypot-Capture-7-1	115	Door Lock	1.00

3.7 Performance Metrics

Performance metrics are important when evaluating classification models. There are several different classification metrics that offer advantages and disadvantages to model assessment. This study considered accuracy, precision, recall, F1 score, the false positive rate and

Matthew's correlation coefficient (MCC). Each of the evaluation metrics offer insights and disadvantages when evaluating classification models.

3.7.1 Accuracy

Accuracy is a naive performance metric that simply offers the ratio of correctly classified observations to total observations. Mathematically accuracy is defined as:

$$Accuracy = \frac{\text{True Positives + True Negatives}}{\text{False Positives + False Negatives + True Positives + True Negatives}} \tag{3.5}$$

Accuracy offers a tractable performance metric, however it can be misleading when considering unbalanced data. For instance, suppose in a labeled dataset consisting of 100 observations, 5 were of class 1 the remaining 95 were of class 2. Then suppose the classifier correctly identified none of class 1 but all of class 2. The classifier accuracy would be .95, but the model failed completely to identify any of class 1. The problem is compounded when there are more than 2 unbalanced classes. This hypothetical problem pertains to malware classifiers where often data is highly unbalanced. In the case of the IoT-23 dataset malicious network flow observations out number the total benign observations. When considering more specific IoT-23 labels some are exceedingly rare. Because of limitations of accuracy other metrics must be considered in conjunction with accuracy.

3.7.2 Precision, Recall, False Postitve Rate, Specificity

Precision and recall help address the problem of evaluating classification models evaluated on unbalanced datasets. Precision is the proportion of positive identifications that were correctly identified and is mathematically expressed as:

$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{3.6}$$

Recall also known as sensitivity is the proportion of actual positives correctly identified and can be mathematically expressed as:

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.7}$$

The false positive rate (FPR) is the proportion of non-relevant or negative cases incorrectly identified as positive cases in the data. It is mathematically expressed as:

$$FPR = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{3.8}$$

This measure is important because false positive classifications present a problem to the end user. Large volumes of false positives limit the capacity of the analyst.

Specificity also known as the true negative rate. Specificity refers to how well a model classifies observations that are not considered the relevant class. It can be mathematically expressed as:

$$Specificity = \frac{\text{True Negative}}{\text{False Positives} + \text{True Negatives}} \tag{3.9}$$

Precision, recall, FPR, specificity are metrics that lend insight into the model. The metrics are non-symmetric. The metrics are dependent upon interpretation of the relevant class. When evaluating malware detection models, expressing each of the metrics in terms of each class provides strengths and weaknesses of the model.

3.7.3 F1 Score

F1 score is a commonly used machine learning metric. It is defined as the harmonic mean of precision and recall. It is succinctly represented mathematically in terms of precision and recall:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.10}$$

F1 scores range in the continuous set of [0,1]. It's important to note since F1 is formulated by definition of precision and recall it is independent from the number of true negatives. Another observation about F1 score is that it is not a symmetric equation. When the relevant class is changed, so does the F1 score. When reporting F1 it's important to analyze

the score in terms of all response classes (Chicco and Jurman 2020). When evaluating malware detection algorithms both classes "true malicious flows" and "true benign flows" are important. Evaluating F1 in terms of all classes provides insight to the effect of response class imbalance. The effect of imbalanced responses can be identified when the F1 score for the majority class and the minority class differ drastically. Multi-class F1 score micro and macro averaging also provide useful interpretations of the F1 score. Micro average accounts for the relative sizes of the response were macro provides an unweighted average.

$$F_1 Micro = \sum_{i=1}^{n} F_{1_i} * \frac{Total_i Observations}{TotalObservations} \tag{3.11}$$

$$F_1 Macro = \sum_{i=1}^{n} F_{1_i} * \frac{1}{n} \tag{3.12}$$

3.7.4 Matthew's correlation coefficient

Matthew's correlation coefficient (MCC) is a symmetric performance measure that accounts for both true and false positives. It can be mathematically expressed as:

$$MCC = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{3.13}$$

The MCC score ranges in the continuous set of [-1, 1], where 1 is a model capable of perfect prediction, -1 the inverse of a perfect model and 0 is equivalent to a model having no predictive capability. MCC is considered an appropriate performance measure when evaluating models subjected to imbalanced response classes.

3.8 Random Forest Modeling

Random Forest is a supervised non-parametric ensemble machine learning algorithm first developed by Breiman (2001). The model creates a multitude of regression or classification trees that each provide input toward a final prediction. In the case of classification each tree is given an equal vote toward determining the predicted response class. The class with the greatest number of tree votes is deemed the predicted class. The algorithm constructs

individual trees by considering all or a subset of the training dataset features. Then the algorithm identifies a root node based upon a feature and value that best splits the data into homogeneous groups. The tree continues to create splits in the data based on the calculation of the Gini Index. The Gini index is an impurity measure that calculates the sum of the squared probabilities for each response class.

$$GiniIndex = 1 - \sum_{j} p_j^2 \tag{3.14}$$

The Gini index score is calculated for each feature in a randomly selected subset of the total number of features. The feature with the lowest Gini index score informs the algorithm of the next node creation and split criteria. The random forest algorithm depicted in Figure 3.1 continues this process independently for each tree in the "so called" forest.

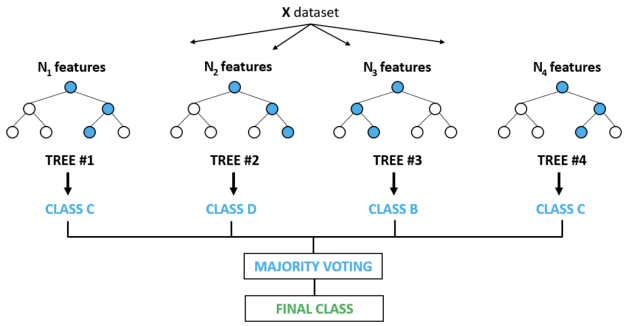

Figure 3.1. Random Forest Diagram. Source: (Abilash 2020).

When training random forest models there is a number of hyper parameters that can be tuned to optimize model performance. The following are the most significant hyper-parameters for this thesis:

- **Number of trees:** The number of trees used in a random forest is an important factor in the predictive accuracy of the method. There is no bias incurred by using a large number of trees, but there is a high computational cost. One method for determining the appropriate number of trees is to calculate the error rate which converges when an optimal number of trees are trained on the data.

- **Tree depth:** The feature limits the number of nodes each tree is able to construct. This is usually limited to prevent over training and to limit computational complexity when training with large datasets.
- **Minimum samples before split:** This feature requires a minimum number of observations for a split node to be created. It is used to limit computational complexity in large datasets and model overfitting.
- **Maximum number of features:** The random forest algorithm considers a maximum number of features before splitting a node. In effect a subset of the feature space is created before tree creation. The default parameter using Python module Scikit-Learn and the R Ranger package is the rounded square root of the total number of features. Other options are possible such as a specified integer or rounded log_2 of the total number of features.
- **Class Weights:** Class weights can be used to penalize the model loss function for falsely classifying minority response classes. This is one of the methods for modeling with imbalanced datasets built into the random forest algorithm.

Random forests are constructed under an assumption that each tree is a "weak learner". An individual tree only understands one aspect of the relationship between a subset of features and the response variable. In the aggregate the trees provide a stronger prediction of the response. There are several advantages to the random forest algorithm. Random forests have been shown to be successful in predictive modeling across a spectrum of applications (Segal 2004). The method also limits the effects of over-fitting through a proper selection of its hyper-parameters (Breiman 2001). Another advantage of random forests that is important for malware detection is that it handles high dimensional data by selecting random subsets of features. The algorithm can be trained in parallel on multiple CPUs which can provide reasonable training times on larger datasets. However, running random forest in parallel comes at a high memory cost that is compounded with big training datasets. We experimented with random forest algorithms using both the R package Ranger and the Python package Scikit-Learn. The NPS high performance computer Grace allowed for access to 24 CPUs and 500 GBs of memory to fit a random forest to the IoT-23 dataset.

One of the disadvantages of random forests is the "black box" nature of the resulting predictor. Although it is possible to interrogate metrics such as feature importance (which we discuss below) and to produce graphical assessments of the marginal effects of predictors,

there is no explicit predictor that can be examined. This limits the ability of an analyst to understand how a prediction is formulated. This drawback, however, is shared by many machine learners including artificial neural networks.

3.8.1 Feature importance

Feature importance calculates the effect of an individual feature in model performance. The Sciket-learn API implements feature importance by calculating a scaled aggregate Gini index score (Pedregosa et al. 2011). It provides tractable results that demonstrate how features contribute to model performance. There are two disadvantages when considering this method. The first is that its scores are calculated on the training dataset and may not generalize to new data. The second is that importances are biased towards continuous features with high cardinality (Pedregosa et al. 2011). Permutation importance is a variation of the concept that overcomes these disadvantages. The latter calculates a baseline score on a fitted model using testing data. It then randomly permutes the values of each feature and recalculates the score. After each iteration permutation importance recalculates the score. The features that most decrease the score after permutation are considered to be the most important (Pedregosa et al. 2011). This method, however, is subject to bias when features are closely correlated (Hooker and Mentch 2019). Using the results of both feature importance and permutation importance offers a balanced approach.

3.9 Deep Neural Network Modeling

The use of DNN models have become widespread in many applications to include malware detection. The architecture of DNNs consists of an input layer of $\boldsymbol{n}$ nodes where $\boldsymbol{n}$ equals the number of features, a series of fully connected hidden layers consisting of activation function nodes, and an output layer (see Figure 3.2). DDN uses gradient descent optimization to estimate the error gradient from observations in the training dataset. Then, through back propagation, the model weights and bias estimates associated with the nodes are updated to reach an optimal solution (Buduma and Locascio 2017).

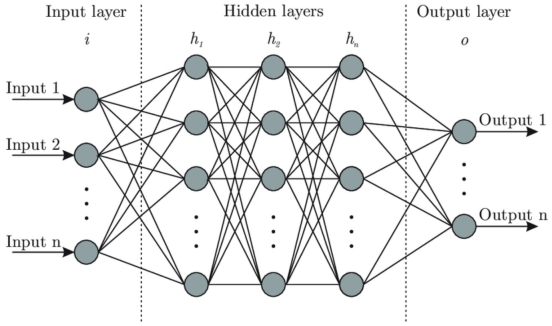

Figure 3.2. Neural Network Diagram. Source: (Shukla 2019).

DNN presents a challenging optimization problem due to the number of hyper-parameters available to the model. The hyper-parameters consist of the following:

- **Number of hidden layers and nodes:** The number of hidden layers, nodes and the connections between, create an exponentially large number (relative to $\boldsymbol{n}$) of trainable weights that are able to be tuned in order to learn non-linear patterns within the dataset. Creating more layers and nodes provides additional weights. Models consisting of just a few layers and nodes are unable to gain insights from highly non-linear patterns, while the opposite can lead to over-fitting of the model. There are heuristics that guide choosing the number of hidden layers but no proven methods. A limiting factor to the number of hidden layers and nodes is computational resources. Advances in the use of graphics processing units (GPU) to train neural networks provide faster training times, allowing for experimentation with larger models. An effective approach is to conduct a series of experiments with the use of cross-fold validation.
- **Activation functions:** Activation functions (depicted in Figure 3.3) compute the weighted sum of inputs and biases to decide if a neuron can be fired or not. The process takes the output signal from inputs and weights from the previous layer and transforms it into a form that can be processed as input to nodes in the next hidden layer. By default, DNNs without specified activation functions map to linear outputs, although activation functions may be non-linear transformations (Buduma and Locascio 2017; Nwankpa et al. 2018).

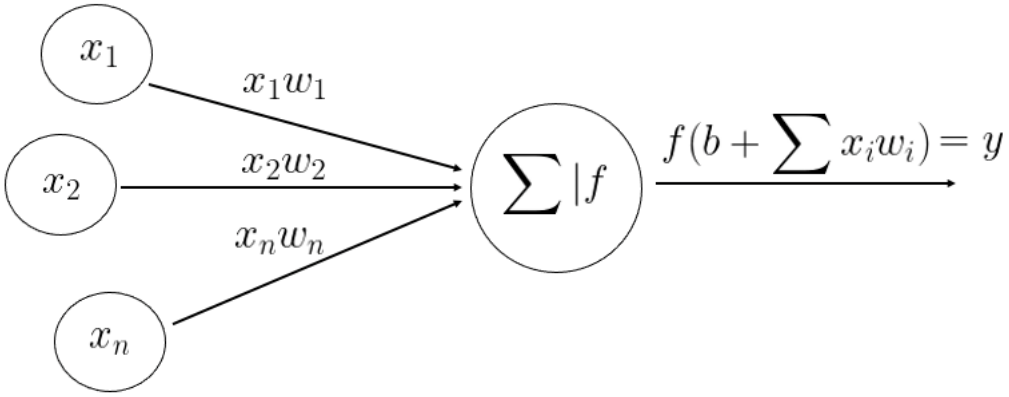

Figure 3.3. Activation Function. Source: (Nwankpa et al. 2018).

Several activation functions are widely used in DNN modeling. The sigmoid function (3.15) is a preferred activation function in the output layer for binary classification because it provides a probability-based outputBuduma and Locascio (2017). The sigmoid function, however, is subject to the vanishing gradient problem, which result in the algorithm failing to properly converge after continued iterations of back propagation (Douglas and Yu 2018).

$$Sigmoid = f(x) = \frac{1}{1 + e^{-x}} \tag{3.15}$$

The rectified linear unit (ReLu) function (3.16) has performed well in many DDN applications and is considered by many practitioners to be the safe default activation function. ReLu is computationally efficient, enabling fast DNN training. The use of this function is subject to the "dying ReLu problem" in which gradient descent fails to alter the weights. Variations such as leaky ReLu and decreasing the learning rate can overcome this phenomenon (Douglas and Yu 2018).

$$ReLu = f(x) = \max(0, x) \tag{3.16}$$

- **Loss functions:** Loss functions are differentiable functions that provide a prediction error term used to inform weights in a DNN. Binary cross entropy loss function (3.17) is the defacto loss function for classification problems consisting of two classes. The loss represents the negative log likelihood providing a probability between [0,1] for

each predicted observation. Binary cross entropy is defined mathematically as:

$$H_y(y) := -\sum_i (y_i \log(y_i) + (1 - y_i) \log(1 - y_i)) \tag{3.17}$$

Binary cross entropy requires that the sigmoid activation function be used for the DNN output nodes. There are some possible yet unpublished alternatives to binary cross entropy, such as modifying F1 or MCC to become a loss function. This practice is not widespread but could possibly provide advantages with imbalanced datasets (Maiza 2019). Keras and Tensorflow allow for the backend development of customized loss functions.

- **Optimizers:** DDN uses algorithms to minimize the output of the loss function by iteratively updating weights and bias estimates in a process known as back propagation. There are several optimization algorithms that use gradient descent and momentum to find a solution. We explored using mini batch stochastic gradient descent (SGD) as presented in the Keras API, and adaptive Moment Estimation (Adam) first introduced by Kingma and Ba (2014). Adam is a memory efficient stochastic optimization process that uses first-order gradients. The algorithm computes individual learning rates for each DNN weight from estimates of the first and second moments of the gradients (Kingma and Ba 2014). Adam converges faster than traditional gradient descent and is widely used in DNN modeling.
- **Learning rates:** Learning rate is a parameter that regulates the amount by which the weights are updated during the training process. Learning rate is calculated as a scalar between [0,1]. A large learning rate allows the gradient descent algorithm to converge quickly but the solution may be suboptimal. On the other hand, learning rates that are small may cause the algorithm to fail to converge. We explored the use of different learning rates to insure optimal performance during DNN training. It is important to note that the Adam optimization algorithm iteratively adjusts the learning rate by multiplying an additional scalar derived from the first and second moments against the learning rate scalar (Kingma and Ba 2014).
- **Class weights:** Class weights can be used to penalize the model's loss function for falsely classifying minority response classes. This is one of the methods for modeling with imbalanced datasets.
- **Number of epochs and batch size:** Batch size is a hyper-parameter that refers

to the number of observations that are fed through the model before updating the model weights. The number of epochs refers to the number of iterations required for the model to incorporate the entire training dataset. DNNs typically need several iterations through the dataset to reach convergence. Adjusting the number of epochs and batch size influences the ability of the model to find an optimal solution. The optimal batch size is dependent upon the circumstances of the data. For this reason, adjusting the number of observations in a batch and number of epochs is important to reaching model convergence.

3.10 Long Short-Term Memory (LSTM) Models

Recurrent neural networks (RNNs) are an adaptation of artificial neural networks that enable learning from sequences of data. Recurrent layers loop data in sequential steps through a node allowing information to be passed from one step of the network to the next . (Buduma and Locascio 2017; Olah 2015). RNNs have been successful in solving speech recognition, language modeling, and translation machine learning problems.

One of the problems with RNNs is that it is prone to an inability to capture data patterns in long sequences in part due the vanishing gradient problem. LSTM networks, introduced by Hochreiter and Schmidhuber (1997), solve this problem by enforcing constant error flow with an efficient gradient based algorithm. One of the key aspects of a LSTM unit is a memory cell that holds information learned sequentially (Patterson and Gibson 2017). With each sequential step the LSTM modifies the memory cell with additional information. Appropriate information is retained by another LSTM component called the "keep gate" (Buduma and Locascio 2017). LSTM algorithms allow for the retention of useful information and the "forgetting" of less important information (Olah 2015).

The same hyper-parameters discussed in section 3.9 that apply to DNNs also apply to LSTMs, but the latter have one additional preprocessing input: the data must be transformed into batches of time sequences of n time steps. Essentially, for each observation this parameter determines the number of past time sequences the model should consider.

We use Python modules Keras and Tensorflow for DNN and LSTM modeling. We use the high performance computing (HPC) frameworks Grace and Hamming provided by NPS.

The HPC resources allow for various combinations of CPU, GPU, and memory based on resource availability. These resources allowed for experimentation of various combinations of hyper-parameters while facilitating timely results.

The advantage of DNN and LSTM modeling is the flexibility to adjust high level aspects of the DNN and achieve excellent predictive results. The disadvantage of DNN and LSTM modeling is the "black box effect" which disallows model inference. These models prevent the analyst from understanding why a particular network flow is predicted malicious or benign. For this reason we use other methods in conjunction with DNNs and LSTMs to gain model inference.

3.11 Stacked Ensemble Modeling

Stacking is simply the use of multiple machine learning models to determine a final prediction. Kaggle competitions are often won with stacked ensemble approaches (Günes 2017). Often stacked approaches use a meta-classifier of either logistic or linear regression that incorporate the predicted outputs of other machine learning algorithms as features to provide a final prediction. We propose the use of the predicted response from random forest as an additional feature included in the training and testing datsets. With the additional feature we train a new DNN model, foregoing the use of a meta-classifier.

$$f_{RF}(X) = \hat{y}_{RF} \longrightarrow f_{DNN}(X \frown \hat{y}_{RF}) = \hat{y}_{DNN} \tag{3.18}$$

This chapter provided a description of the IoT-23 dataset, the proposed methods for cleaning and feature engineering. The chapter provided performance measures important to classification models trained with imbalanced datasets. The chapter also highlighted important aspects of the proposed machine learning algorithms we use in this thesis for malware detection with a emphasis on key hyper-parameters for performance optimization.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Results and Analysis

Chapter 4 presents key aspects of our findings. We start with an analysis of tree-based classification and inference models. We find that classification decision trees and random forests provide useful insights concerning feature importance, model robustness, and prediction. Next, we discuss the findings of our traditional deep neural network models, the stacked models and finally our LSTM models. We identify an LSTM model that is well-suited for botnet detection due to the temporal nature of botnet activity.

4.1 Decision Tree Classification

Our decision tree classification model balances the dataset with the under-sampling technique. This method reduces the number of malicious network data flows to balance the response class ratio. Using five-fold cross-validation we then explore a series of hyperparameter combinations. The hyperparameters that provide the best performance are: limiting the maximum tree depth to 6, the maximum number of leaf nodes to 10, and setting the minimum impurity decrease to 0.001. Our best model achieves an accuracy of 0.972, MCC 0.94 and F1 scores of 0.97 with the test dataset (MCC and F1 defined in Chapter 3.7). This result indicates strong model performance. A summary of model performance is given in Tables 4.1 and 4.2, and Figure 4.1.

Table 4.1. Binary DT Performance Metrics

	Metric	Value
1	Accuracy	0.972
2	MCC	0.944
3	FPR	0.040

Table 4.2. DT Precision, Recall, F1

	precision	recall	F1	support
Benign	0.98	0.96	0.97	53757
Malicious	0.96	0.98	0.97	54045
Maro avg	0.97	0.97	0.97	107802
Weighted avg	0.97	0.97	0.97	107802

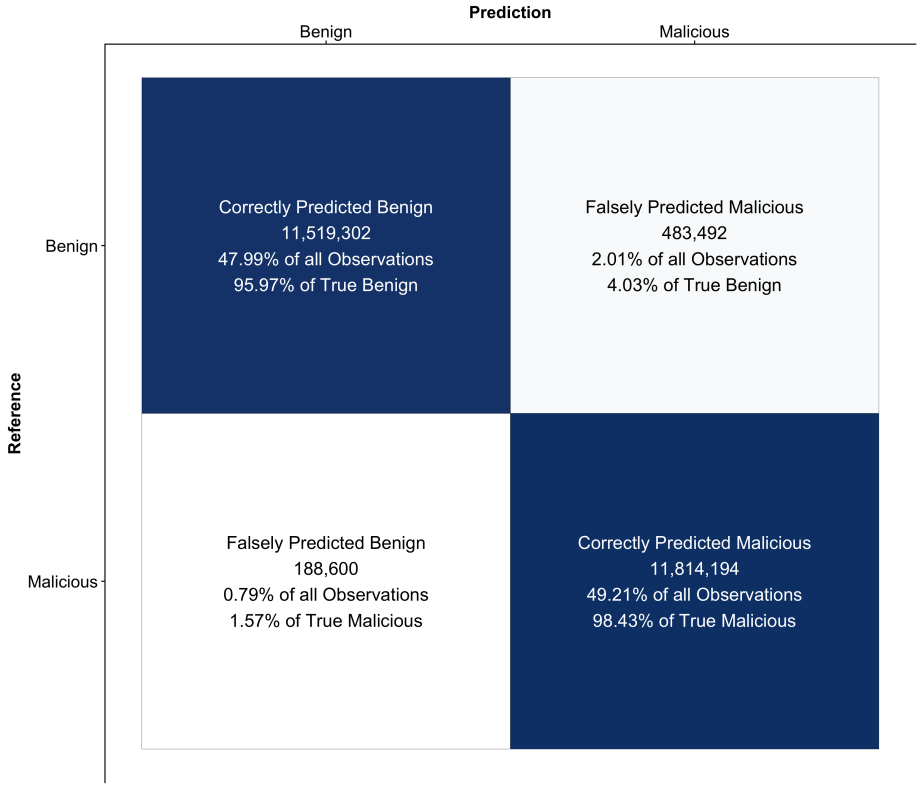

Figure 4.1. DT Confusion Matrix

The decision classification tree renders useful diagrams (Figure 4.2) that provide insight for malware detection. The model indicates that the features associated with well-known ports from the destination source, and the number of flows in the past 0.1 second accounts for much of the model variance.

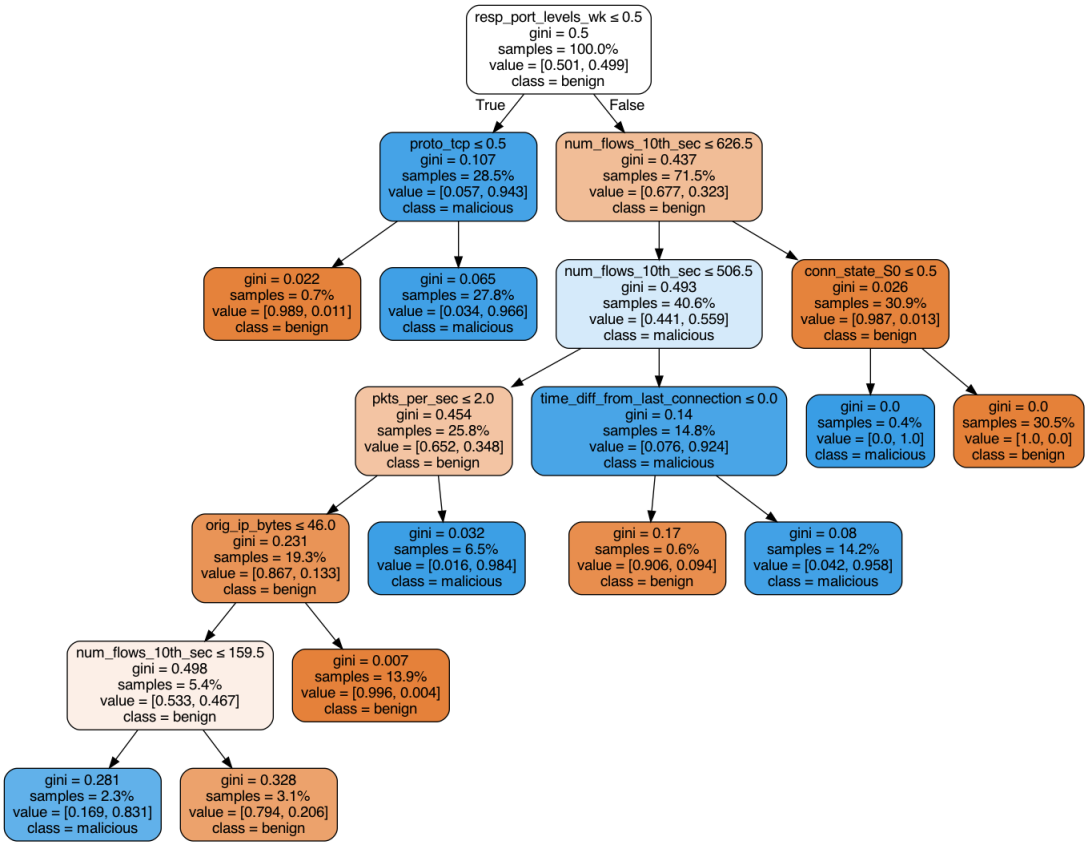

Figure 4.2. Decision Classification Tree

We then use decision classification trees to create models for each of the IoT-23 detailed response labels. The purpose of this modeling is to provide inference for the different behaviors demonstrated by malware botnets. Specific models were created for C&C, attack, port scanning, and DDoS labeled traffic. Each of the subsets were filtered to include only benign and malware behaviors of interest. For each of these models we again under-sample the majority class in order reduce model bias. For each DT model we explore various parameters through cross validation.

4.1.1 C&C Server Decision Tree Classification

For our C&C traffic model we consolidate all C&C variants into one global C&C label from the training dataset. We then take C&C traffic and divide it further into another training and test partition. The purpose of the partition is to preserve our global testing

data for future model test set evaluation. Through cross validation, the C&C server traffic model performs best by limiting tree maximum depth to 5 and the number of leaf nodes to 9. The model demonstrates high predictive accuracy as shown in the summary of model performance represented in Tables 4.3, 4.4 and Figure 4.4. We observe from Figure 4.3 that the differences between consecutive network flows, the packets per seconds, and the protocol not being TCP traffic are able to account for much of the model variance.

Table 4.3. DT C&C Server Traffic

	Metric	Value
1	Accuracy	0.990
2	MCC	0.980
3	FPR	0.012

Table 4.4. DT C&C Server Traffic Precision, Recall, F1

	precision	recall	F1	support
Benign	0.99	0.99	0.99	9948
Malicious	0.99	0.99	0.99	9948
Maro avg	0.99	0.99	0.99	19646
Weighted avg	0.99	0.99	0.99	19646

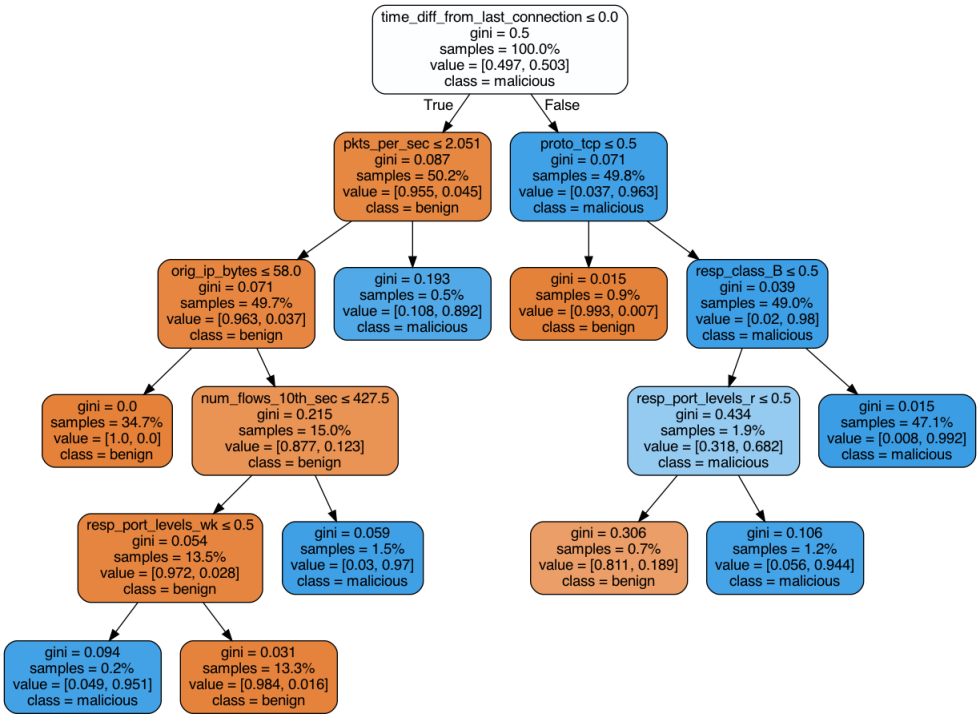

Figure 4.3. DT for C&C server

Figure 4.4. DT C&C Server Traffic Confusion Matrix

4.1.2 Attack Traffic Decision Tree Classification

Attack traffic accounts for a relatively small portion of the dataset with only 9,398 network flows belonging to the class. Attack traffic is primarily distributed between the Muhstik and Mirai botnet captures. Interestingly, the attack flows in the dataset appear to be very homogeneous. As witnessed in Figure 4.5 simply splitting the destination to source transaction bytes by less than or equal to 78 the model correctly classifies 99.9% of the samples. This likely indicates that the attacks represented in this dataset only represent a very specific example and not generalize to other attacks. However, it does highlight a possible opportunity for malware detection in IoT devices. IoT devices with specific purposes often exhibit characteristic network traffic patterns. Therefore, simple rules developed by tools such as decision trees may be effective in detecting malware infection for those devices.

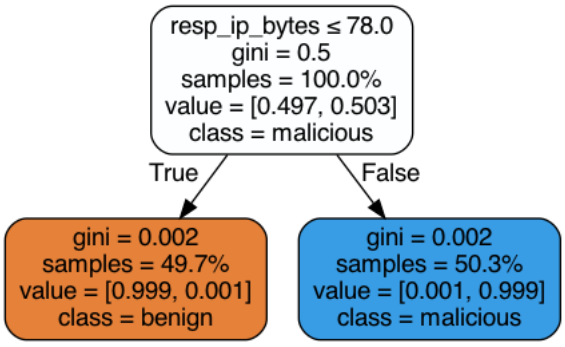

Figure 4.5. DT Classification for Attack Traffic

4.1.3 DDoS Traffic Decision Tree Classification

To model DDoS we again create a balanced partition of benign and DDoS labeled network flows. Through cross-validation the model performs best by limiting the maximum tree depth to 4, the maximum number of leaf nodes to 8, and the minimum impurity decrease to 0.001. From Figure 4.6 we see that the DDoS model root node split is based upon the destination port being within the dynamic port range (ports 49152-65535). Model performance presented in Tables 4.5 and 4.6 and in Figure 4.7 demonstrates strong model performance. However, nearly all of the DDoS traffic is correctly classified simply from the flow association TCP in the dynamic port range. This condition is unlikely to generalize to a wider range of IoT devices with unique applications. Many normal applications use TCP in combination with dynamic ports.

Table 4.5. DT DDoS Traffic

	Metric	Value
1	Accuracy	0.999
2	MCC	0.998
3	FPR	0.002

Table 4.6. DT DDoS Traffic Precision, Recall, F1

	precision	recall	F1	support
Maro avg	0.998	0.998	0.998	351665
Weighted avg	0.998	0.998	0.998	351665

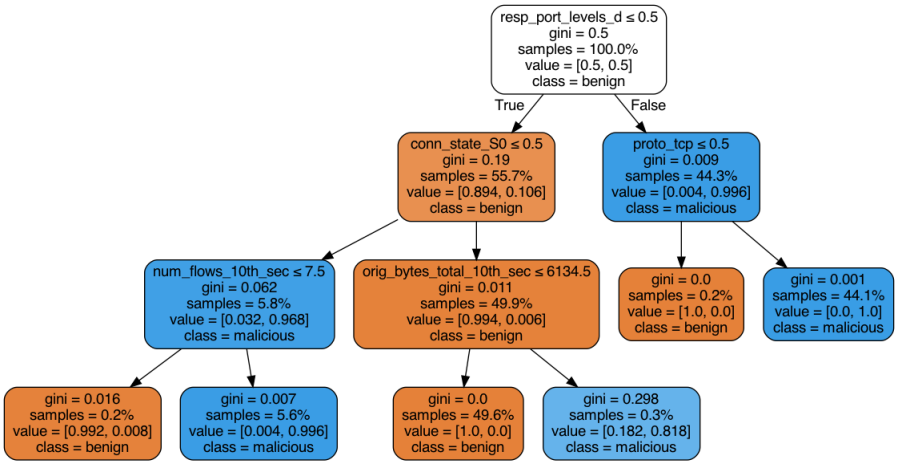

Figure 4.6. DDoS Traffic DT Classification

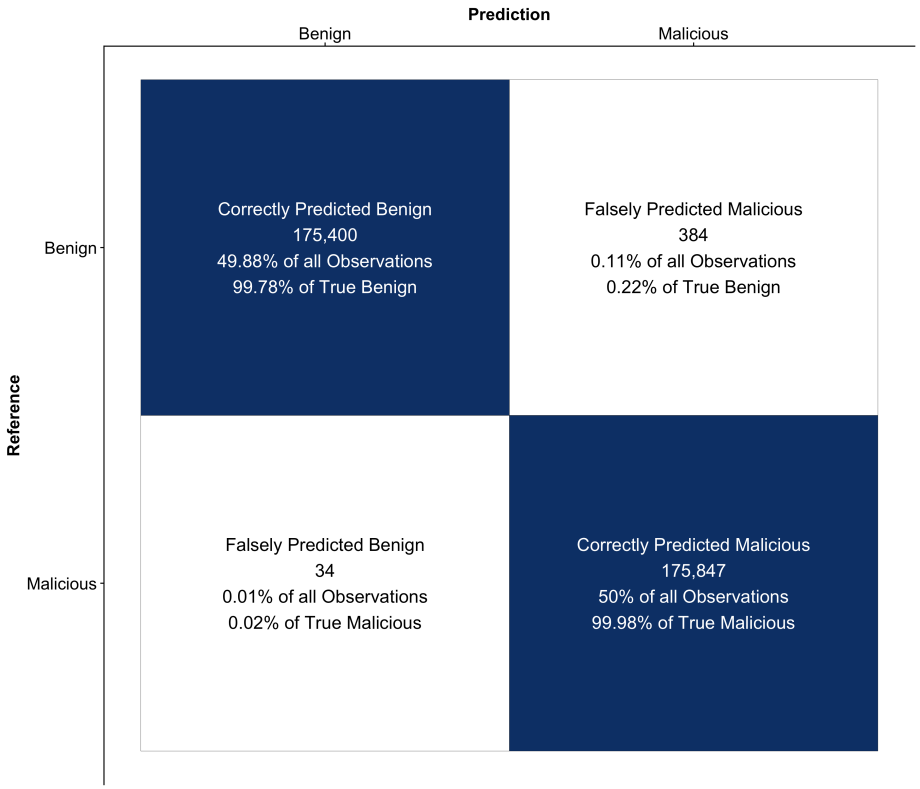

Figure 4.7. DT DDoS Traffic Confusion Matrix

4.1.4 Port Scanning Traffic Decision Tree Classification

We randomly select a balanced subset from the training data of benign flows and flows labeled part of a horizontal port scan. Compared to our other tree models the port scanning model is the most complex. The complexity is demonstrated in tree depth and the large number of leaf nodes as shown in Figure 4.8. Using cross validation, the model was limited to a maximum tree depth of 9, a maximum number of leaf nodes of 14, and minimum impurity decrease set to 0.001. Model accuracy and MCC is lower compared to our other models. This indicates horizontal port scanning is more challenging to detect with independent network flow observations. Additional model performance metrics are represented in Tables 4.7, 4.8 and Figure 4.9. This class is also the most represented in the IoT-23 dataset and likely is the source of most model error. The feature "number of flows in the past 10^{th} of a second" accounts for many of the partitions in the tree. This indicates an important time dependence aspect for classifying port-scanning behavior.

Table 4.7. DT Port Scanning Traffic

	Metric	Value
1	Accuracy	0.976
2	MCC	0.953
3	FPR	0.030

Table 4.8. Port Scanning Traffic Precision, Recall, F1

	precision	recall	F1	support
Benign	0.99	0.97	0.98	270030
Malicious	0.97	0.98	0.98	269747
Maro avg	0.98	0.98	0.98	539777
Weighted avg	0.98	0.98	0.98	539777

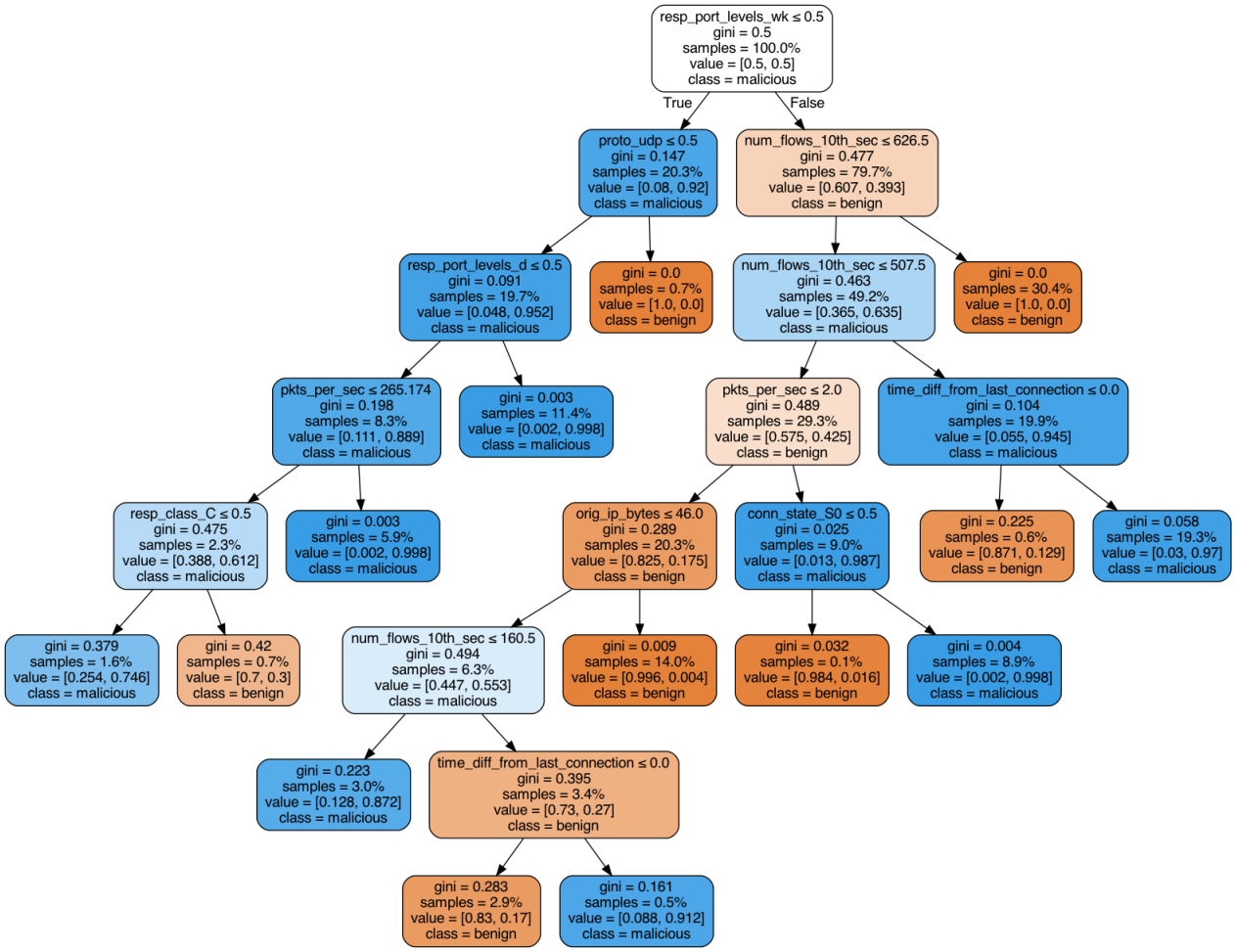

Figure 4.8. Port Scan Traffic DT Classification

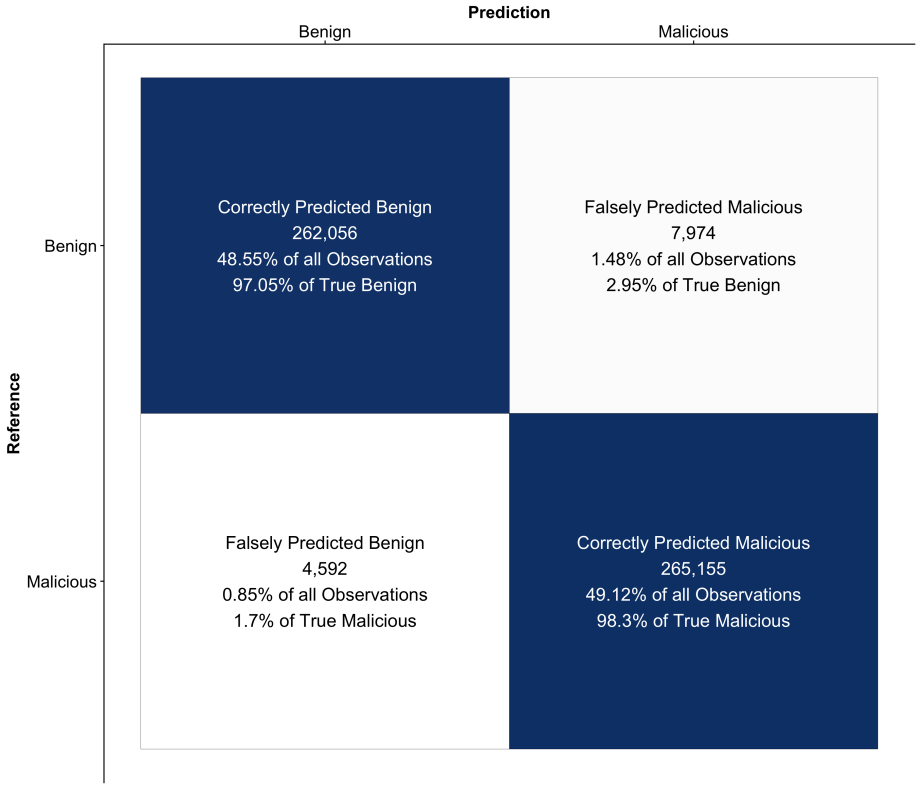

Figure 4.9. DT Port Scan Traffic Confusion Matrix

4.2 Random Forests

With our random forests models, we again balance the data by down sampling malicious network flows. We find that neither oversampling the minority class nor SMOTE (defined in Chapter 3) performs better than the down-sampling technique. In order to determine the best parameters, we use a one percent sample of the training data for the purpose of cross-fold validation with different hyperparameters. The best model parameters obtained from this analysis limited the depth to 15 nodes per tree, and the maximum number of features to the square root of the total number of features per tree. We find that 100 trees is sufficient to achieve near optimal model performance. We validate the model parameters by retraining the model with the global training dataset and evaluating model performance with the global test dataset. When increasing the volume of data, we find that accuracy increases by approximately half a percentage point. Our random forest model performance is stronger

than the single decision tree model in regards to all metrics. Notably, there is a 0.53% reduction in the ratio of false positive classification. A summary of model performance is presented in Tables 4.9, 4.10 and Figure 4.10.

Table 4.9. Random Forest Performance Metrics

	Metric	Value
1	Accuracy	0.980
2	MCC	0.959
3	FPR	0.03

Table 4.10. Random Forest Precision, Recall, F1

	precision	recall	F1	support
Benign	0.99	0.97	0.98	12,002,794
Malicious	0.97	0.99	0.98	12,002,794
Maro avg	0.98	0.98	0.98	24,005,588
Weighted avg	0.98	0.98	0.98	24,005,588

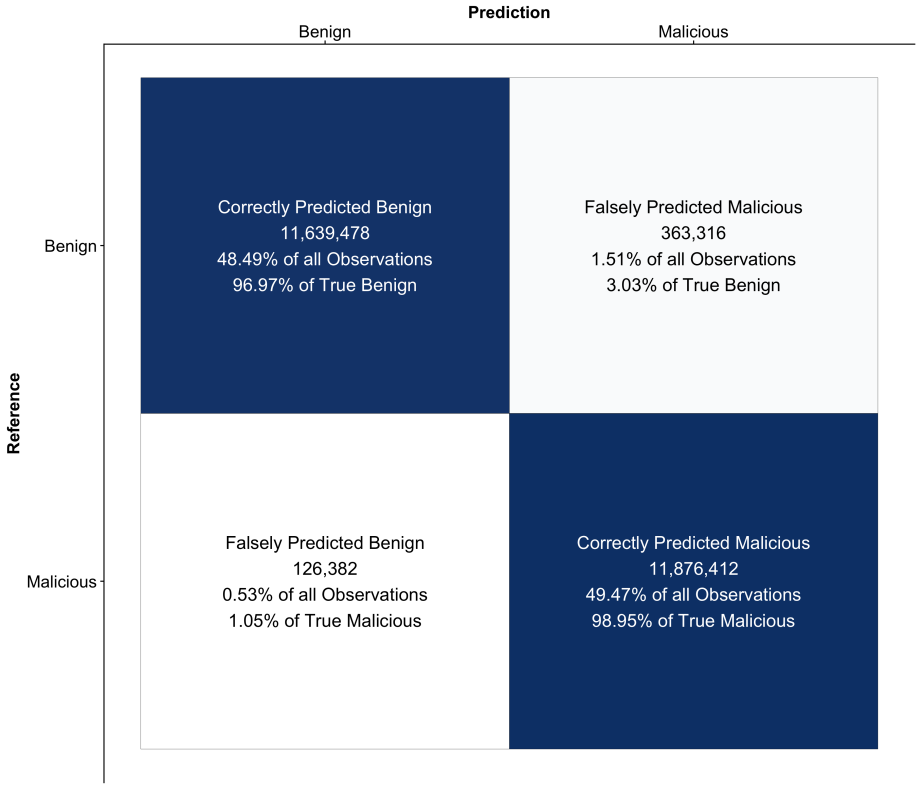

Figure 4.10. Random Forest Confusion Matrix

We use feature importance metrics to identify predictors that contribute most to model performance. To validate feature importance we compare our results to permutation importance and find no significant differences. The most important features are: the number of flows in the past 10^{th} of a second, the destination port association with well-known ports, and the time difference between consecutive flows. Unsurprisingly, the results of this analysis agree closely with results obtained from the decision tree classification models. These results indicate that network flows are time dependent. Figure 4.11 provides visualization of the most important features for the random forest model.

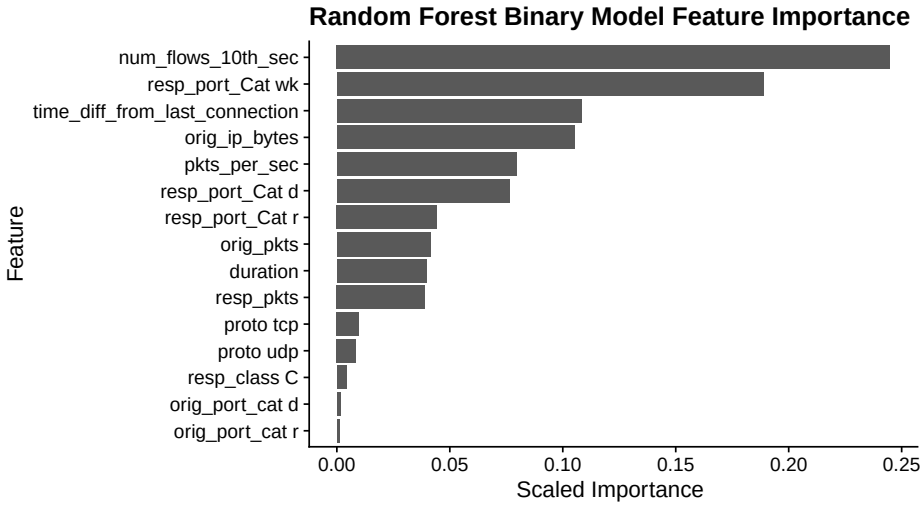

Figure 4.11. Random Forest Binary Model Feature Importance

4.3 Deep Neural Network Model

We find that traditional DNN models do not perform as well as tree-based models. Again, we find that down-sampling the majority class provides more robust model performance. We obtain higher MCC and accuracy scores with less balanced data but at the cost of an undesirable false positive rate. After experimentation we ultimately find the standard scaler to offer better performance than other scaling techniques. After several trials we settle on a five-layer model containing 67,48,34,16, and 1 nodes per respective layer. We find the relu activation function followed by the sigmoid function on the output layer to produce the best performing model. Modest improvements result when adding dropout layers between each hidden layer for feature regulation. A summary of the best DDN model performance is presented in Tables 4.11, 4.12 and Figure 4.12.

Table 4.11. Deep Neural Network Performance Metrics

	Metric	Value
1	Accuracy	0.959
2	MCC	0.918
3	FPR	0.048

Table 4.12. Deep Neural Network Model Precision, Recall, F1

	precision	recall	F1	support
Benign	0.97	0.95	0.96	12,002,794
Malicious	0.95	0.97	0.96	12,002,794
Maro avg	0.96	0.96	0.96	24,005,588
Weighted avg	0.96	0.96	0.96	24,005,588

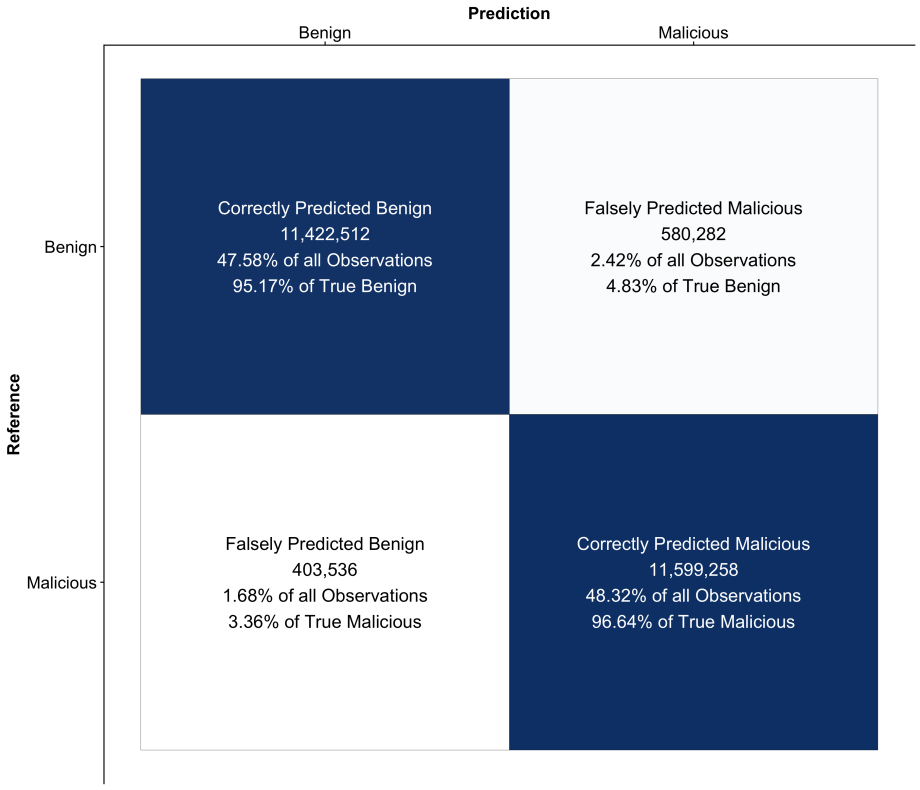

Figure 4.12. DNN Model Confusion Matrix

4.4 Stacked Model

Our stacked model uses a similar structure to our DDN model. As an added feature the stacked model includes the predicted values from the random forest model. Our results find that the stacked model performs slightly better than our random forests model as indicated

by its MMC (.965) and FPR (.023) scores. Additional stacked model performance metrics are presented in Tables 4.13, 4.14 and Figure 4.13.

Table 4.13. Stacked Model

	Metric	Value
1	Accuracy	0.983
2	MCC	0.965
3	FPR	0.023

Table 4.14. Stacked Model Precision, Recall, F1

	precision	recall	F1	support
Benign	0.99	0.98	0.98	12,002,794
Malicious	0.98	0.99	0.98	12,027,794
Maro avg	0.98	0.98	0.98	24,005,588
Weighted avg	0.98	0.98	0.99	24,005,588

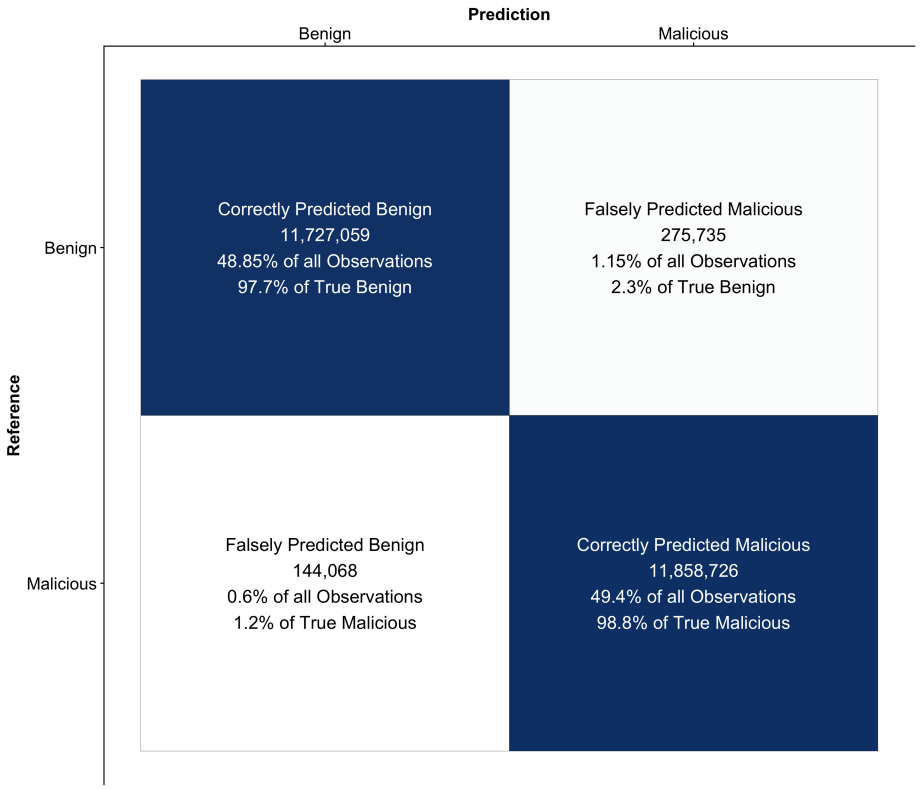

Figure 4.13. Stacked Model Confusion Matrix

4.5 LSTM Model

Our analysis of decision trees and random forest models demonstrates the temporal nature of malware botnet traffic in IoT devices. LSTMs are well suited for prediction problems with time-based dependencies. Our methods for LSTM modeling differ from our other models in important respects. For this analysis we use an aggregated version of the entire IoT-23 dataset, which we reduce from 300 million to 2.3 million network flows by aggregating in time windows of 10^{th} of a second. Much of the quick succession network traffic generated by malware port scans and DDoS attacks is condensed within small time periods. The aggregation process calculates summary statistics (sum, mean, standard deviation) for each feature within each time window. We find the aggregation process important to LSTM modeling because of the computational complexity when transforming data into three dimensional arrays required by LSTM models and the memory cost incurred from the transformation.

For LSTM modeling we adopt the framework developed for the classification of human activity(Zebin et al. 2018; Ordóñez and Roggen 2016). We find that our classification prob-lem shares similarities with this framework. Most notably, the human activity classification problem relies upon samples from different humans during different time periods. Simi-larly, our botnet detection model uses training data from different devices, at different time periods, and with unique botnet malwares.

Our LSTM model requires the data to be in sequences of 10 observations. We use a time step length of 10 as the rolling start point of each consecutive time interval sequence. The process creates 234,510 time intervals from the 2.3 million aggregated observations. In essence, for each consecutive step (10 observations) the model takes into account the feature values up to ten periods back in time. We designate the response variable by the most frequent class mode within each of the time steps. We apply this process to a partition of the data used for model training and to another partition designated model evaluation. The testing data consists of following IoT-23 data scenarios: CTU-Honeypot-Capture-7-1, CTU-IoT-Malware-Capture-49-1, CTU-IoT-Malware-Capture-60-1, and CTU-IoT-Malware-Capture-7-1.

The model structure uses the Keras sequential model API. We use a bidirectional wrapper over the Keras LSTM input layer with 128 units, then add a dropout layer with rate 0.5, a 128 unit dense layer with a relu activation function, an output layer with a softmax activation function. We find the Adam optimizer and categorical cross-entropy to provide the best model performance. We also find with batch size 32 the model converges using early stopping at the 73^{rd} epoch.

We find the LSTM model performance to be superior to our other models. This is best demonstrated by the confusion matrix. The model correctly identifies all 53 of the benign sequences while incorrectly classifying only 18 of the 18,704 malicious sequences. We are, however, unable to create a balanced aggregated dataset without corrupting the temporal nature of the data. It is important to note that MCC is disproportionately skewed by the small number of false negative classifications. The lower than expected MCC, benign-precision is attributed to the low number of benign observations. The model performance is especially promising because we introduce a new non-infected IoT device (Amazon Echo) and new botnet variants (Gagfyt) to the testing data. Overall, the LSTM model

shows promise for malware botnet classification. A summary of model performance is represented in Tables 4.15, 4.16 and Figure 4.14.

Table 4.15. LSTM Model

	Metric	Value
1	Accuracy	0.999
2	MCC	0.863
3	FPR	0.000

Table 4.16. LSTM Model Precision, Recall, F1

	precision	recall	F1	support
Benign	0.75	1.0	0.85	53
Malicious	1.0	1.0	1.0	18,722
Maro avg	1.0	1.0	0.93	18,775
Weighted avg	1.0	1.0	1.0	18,775

Figure 4.14. LSTM Model Confusion Matrix

4.6 Summary of Model Performance

The models indicate that detecting various malware botnets is possible with the use of machine learning. High accuracy with balanced datasets indicate that our models and features enable the detection of malicious activity in the IoT-23 dataset scenarios.

One of the most interesting findings was the success found in the decision tree classification models. We find decision tree classification particularly beneficial for detecting C&C server traffic, attacks, and DDoS. Some of the success is likely contributed to the homogeneous nature of the network traffic in the IoT-23 dataset. The result is promising because the analysis of both specific IoT devices and the general behaviors of malware botnets indicate that network flow based rules deriving from decision tree classification can provide indication that an IoT device has been compromised. A rules-based approach would be especially beneficial for IoT devices that have specific functionality and a limited range of benign

network traffic activity.

Through model inference we find our features based upon time, and features engineered with domain knowledge aid model performance. The counting of number of flows in the 10^{th} of a second time window at each flow is especially useful for capturing the temporal nature of botnet activity.

We discover the LSTM model performs the best compared to other models due to the temporal nature of botnet activity. The model also indicates that data aggregation and time series-based models can accurately detect botnet activity.

CHAPTER 5:
Conclusion

5.1 Review of Study Questions

In this chapter we revisit the study questions that we presented in Chapter 1 and explain how our research findings address them. We also discuss the limitations and contributions of our work to broad efforts to detect malicious activity in DoD networks comprised of diverse components. Finally, we describe future directions for research that we hope our research will cultivate.

5.1.1 Which machine learning algorithms are well suited for malware detection using network flow data?

We find that supervised machine learning classification algorithms are promising for malware botnet detection. Bidirectional LSTM models perform best in our research context due to the temporal nature of botnet traffic. Specifically, we propose the use of aggregated Zeek connection logs to train and deploy LSTM models for malware detection. Zeek greatly reduces the number of observations compared to PCAP, although Zeek still produces large volumes of data even for small networks. Our proposed LSTM model in conjunction with data aggregation of network flows reduces the number of observations. Data aggregation is useful if the client is not less concerned with determining which network connections are malicious than if a particular IoT device is infected with botnet malware. The incurred classification delay is at least 2.5 seconds plus computational overhead with the proposed step size and data aggregation windows of a 10^{th} of second. This may be sufficient for network security operators. The step size and aggregation time window can be toggled for specific network circumstances and still achieve good classification performance.

Classification trees can provide valuable information about the activity of botnets. We also find that classification trees achieve high classification accuracy on balanced datasets to detect C&C server, attack and DDoS network traffic. The proposed decision tree classification models and random forests models may benefit an automated botnet detector by providing a

network flow classification once our proposed LSTM model detects an infected IoT device within a time window. The specific network flow classifications may provide a more refined understanding to network security operators.

5.1.2 Which data features are best suited for machine learning malware botnet detection?

We propose a number of engineered features deriving from Zeek connection logs. The time lag features, and time difference features are valuable for botnet malware detection due to the temporal nature of botnet traffic. Port level features indicating well-known, registered and dynamic ports are also important features indicated by our proposed tree-based classification models. Tree models, variable importance derived from Gini index scoring and high test set accuracy consistently demonstrate our engineered features are valuable for classification of botnet infected IoT devices.

5.1.3 Are machine learning algorithms trained with the Iot-23 data set generalizable for wider use?

Machine learning models developed solely with the use of the IoT-23 dataset will likely produce a high volume of false positive classifications when exposed to new IoT network traffic. The lack of diversity of IoT devices in the IoT-23 dataset prevents machine learning models from generalizing to a wide range of IoT network traffic. The evidence supporting our assessment is best demonstrated in section 4.1.2 where our decision classification tree model for attack traffic classified 99.99% of all labeled network flows based only on the destination source sending greater than 78 bytes per network flow connection. There are some IoT devices for which this could be a normal occurrence. The IoT-23 dataset captures a wide array of IoT botnets, although the botnet behavior is relatively homogeneous across variants. The temporal patterns of DDoS and port scanning would likely generalize to other networks. We expect some degradation of performance relative to our findings because the tempo of DDoS and port scanning are functions of both network bandwidth and computational power of the IoT device . Our proposed models would benefit from being retrained with additional data specific to IoT devices and networks of interest.

5.2 Limitations

Our analysis considered only the IoT-23 dataset, which is limited to only a few different IoT devices. The strength of the IoT-23 dataset is the wide variety of malware botnet variants that it contains. A dataset with more benign and diverse IoT network traffic would increase model performance in a more general context .

Our analysis did not explore the case of long connections. Zeek normally logs connections after the connection is complete. Typical connections in the IoT-23 dataset last less than a second. However, some connections associated with unique applications can last several hours. There are open source Zeek packages that allow for longer connections to be logged differently. More work should be conducted in order to understand the effects of long connections and machine learning models.

We did not explore all forms of malware. We focused our study primarily on botnets found in IoT devices. Botnets are not the only threats to those devices. Some of the methods we propose may be adapted to new threats. Because botnets generate many network flow connections, malware that can propagate and attack with fewer connections may present a greater challenge for detection.

5.3 Contributions

Our work makes several contributions to the development of statistical methods for malware detection, including the use of time-lag features, the development of inferential tree based classification models, and the development of a bidirectional LSTM model with aggregated Zeek connection logs. Although other investigators have used the IoT-23 dataset, we are unaware of any who have trained models that encompass all IoT-23 malware captures.

There is a great deal of published research on applying machine learning algorithms to network flow malware classification. This corpus employs a wide array of algorithms that demonstrate high accuracy and other desirable performance metrics. But there has been little work that explores these methods in the context of botnet classification. Our use of decision classification trees allows us to understand the limitations of our training data. It also exposes possible opportunities to detect malware. We demonstrate that relatively shallow trees are able to classify, with high accuracy, malware variants that include C&C,

DDoS and attacks on IoT devices. From the decision tree splits logical network rules can be developed for specific IoT devices on the network. For example, an IoT device may have no legitimate reason to receive packets and bytes above a certain threshold. These thresholds can be derived from decision tree classification models. Simple rules derived from our proposed models may help network security analyst detect malware infected IoT devices.

We demonstrate the benefit of time lag variables in network flow data for botnet malware detection. Our time lag variables enhance traditional machine learning algorithms that are based on the assumption of independent observations. Through our inferential models and random forest variable importance metrics we demonstrate the importance of exploiting temporal features.

Our bidirectional LSTM model is unique in that it achieves high accuracy with the use of aggregated Zeek connection logs. This aggregation technique greatly reduces the number of observations without loss of generality. It is important because we are thus able to train our LSTM model using 23 different benign and malicious scenarios with less computational cost. Our approach allows LSTM models to capture a wide variety network traffic with fewer observations.

5.4 Future Work

We propose that researchers create a small network lab in order to study the behavior of malware infected IoT devices. With minimal hardware and the availability of malware open-source code there is tremendous opportunity for continued study. The ability to create new datasets would allow malware scenarios to be simulated that pertain to unique DoD devices and applications. There are many intricacies to network traffic that would become more transparent through custom development. This would allow researchers to explore in greater depth machine learning methods that generalize to specific networks of interest.

Our proposal for data aggregation techniques reduces the number observations but increases the number of features. We did not fully explore the use of dimensionality reduction techniques such as principal components or deep auto encoders. Dimensionality reduction methods may have the potential to increase the efficiency of model development as datasets increase in size.

Our work also did not explore the use of unsupervised or semi supervised learning techniques. Unsupervised learning may provide benefits independently or in conjunction with our proposed supervised learning techniques. Unsupervised clustering techniques may also help aid additional feature engineering. A special clustering analysis of the Zeek connection log field "history" may be fruitful.

Finally, we suggest that future work be devoted to creating a user interface that could leverage the machine learning models we propose in this thesis. This may entail integrating some of our proposed feature engineering techniques within the Zeek framework. Another important aspect would be to consider how false positive classifications would inform decision rules and potentially update classification models or the program's logic. All models are inherently wrong, the question is how can machine learning malware detectors become especially useful to the end user?

THIS PAGE INTENTIONALLY LEFT BLANK

References

Abilash R (2020) Applying random forest (classification) machine learning algorithm from scratch with real datasets. *Machine Learning*. Accessed April 13, 2021, https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57.

Akub Kroustek ASJN Vladislav Iliushin, Hron M (2018) New, more sophisticated IoT botnet targets a wide range of devices. *Avast Blog*. Accessed April 29, 2021, https://blog.avast.com/new-torii-botnet-threat-research.

Breiman L (2001) Random forests. *Machine Learning* 45(1):5–32.

Buduma N, Locascio N (2017) *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms* (O'Reilly Media, Inc., Sebastopol, CA).

Bursztein E (2017) Inside Mirai the infamous IoT botnet: A retrospective analysis. *Elie Bursztein's Site*. Accessed, March, 31, 2021, https://www. elie. net/blog/security/inside-mirai-theinfamous-iot-botnet-a-retrospective-analysis.

Chawla N, Bowyer K, Hall L, Kegelmeyer P (2002) SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* 16:321–357.

Chawla NV, Japkowicz N, Kotcz A (2004) Special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter* 6(1):1–6.

Chellel K (2019) The hacker who took down a country. *Bloomberg*. Accessed April 29,2021, https://www.bloomberg.com/news/features/2019-12-20/spiderman-hacker-daniel-kaye-took-down-liberia-s-internet.

Chicco D, Jurman G (2020) The advantages of the Matthews Correlation Coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21(1):1–13, Accessed April 29, 2021, https://doi.org/10.1186/s12864-019-6413-7.

Douglas S, Yu J (2018) Why relu units sometimes die: analysis of single-unit error back-propagation in neural networks. *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 864–868 (IEEE).

Dutta V, Choraś M, Pawlicki M, Kozik R (2020) A deep learning ensemble for network anomaly and cyber-attack detection. *Sensors* 20(16):4583.

Gibert D, Mateu C, Planes J (2020) The rise of machine learning for detection and classification of malware: research developments, trends and challenges. *Journal of Network and Computer Applications* 153:102526.

Goralski W (2017) *The illustrated network: how TCP/IP works in a modern network* (Morgan Kaufmann, Cambridge, MA).

Grégio ARA, Afonso VM, Filho DSF, Geus PLd, Jino M (2015) Toward a taxonomy of malware behaviors. *The Computer Journal* 58(10):2758–2777.

Günes F (2017) Why do stacked ensemble models win data science competitions. *SAS Institute Blog* .

Gustavsson V (2019) *Machine Learning for a Network-based Intrusion Detection System: An application using Zeek and the CICIDS2017 dataset*. Bachelor thesis, Royal Institute of Technology, KTH School of Electrical Engineering and Computer Science, Stockholm, Sweden, https://www.diva-portal.org/smash/record.jsf?pid=diva2:1324795.

Hale J (2019) Scale, standardize, or normalize with Scikit-Learn. *Towards Data Science.* Accessed February 16, 2021, https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02.

Harbert T (2017) Practical uses of the internet of things in government are everywhere, government technology. *Govtech.* Accessed April 29, 2021, https://www.govtech.com/network/Practical-Uses-of-the-Internet-of-Things-in-Government-Are-Everywhere.html.

Herwig S, Harvey K, Hughey G, Roberts R, Levin D (2019) Measurement and analysis of hajime, a peer-to-peer iot botnet. *Network and Distributed Systems Security (NDSS) Symposium*.

Herzbeg B, Zeifman I, Bekerman D (2016) Breaking down mirai: An IoT DDoS botnet analysis. *Imperva.* Accessed April 29, 2021, https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet.

Hilt S, Mercês F, Rosario M, Sancho D (2020) Worm war: The botnet battle for IoT territory. Accessed April 29, 2021, https://documents.trendmicro.com/assets/white_papers/wp-worm-war-the-botnet-battle-for-iot-territory.pdf.

Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Computation* 9(8):1735–1780.

Hooker G, Mentch L (2019) Please stop permuting features: An explanation and alternatives. *arXiv preprint arXiv:1905.03151* Accessed April 29, 2021, https://arxiv.org/pdf/1905.03151.pdf.

Horneman A, Dell N (2014) Smart collection and storage method for network traffic data. Technical report, Carnegie Mellon University Software Engineering Institute, Hanscom AFB, MA, https://resources.sei.cmu.edu/asset_files/TechnicalReport/2014_005_001_304866.pdf.

Huang W, Song G, Li M, Hu W, Xie K (2013) Adaptive weight optimization for classification of imbalanced data. *International Conference on Intelligent Science and Big Data Engineering*, 546–553 (Springer, New York, NY).

Hussain F, Abbas SG, Fayyaz UU, Shah GA, Toqeer A, Ali A (2020) Towards a universal features set for IoT botnet attacks detection. *ArXiv Labs Cornell University.* Accessed April 29, 2021, https://arxiv.org/abs/2012.00463.

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *ArXiv Labs Cornell University.* Accessed 20, April, 2021, https://arxiv.org/abs/1412.6980.

Kolias C, Kambourakis G, Stavrou A, Voas J (2017) DDoS in the IoT: Mirai and other botnets. *Computer* 50(7):80–84.

Maiza M (2019) The unknown benefits of using a soft-F1 loss in classification systems. *Towards Data Science.* Accessed April 29, 2021, https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d.

Meidan Y, Bohadana M, Mathov Y, Mirsky Y, Shabtai A, Breitenbacher D, Elovici Y (2018) N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing* 17(3):12–22.

Meneghello F, Calore M, Zucchetto D, Polese M, Zanella A (2019) IoT: Internet of threats? a survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal* 6(5):8182–8201.

Musil S (2016) Internet outage in germany blamed on failed botnet attack. *Cnet.* Accessed March 16, 2021, https://www.cnet.com/news/internet-outage-in-germany-blamed-on-failed-botnet-attack.

Naeem H, Guo B, Naeem MR (2018) A light-weight malware static visual analysis for IoT infrastructure. *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 240–244 (IEEE).

Newman L (2017) What we know about friday's massive east coast internet outage. *Wired Magazine.* Accessed, April 15, 2021, https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn.

Nwankpa C, Ijomah W, Gachagan A, Marshall S (2018) Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378.* Accessed 20 April 2021, https://arxiv.org/pdf/1811.03378.pdf.

Olah C (2015) Understanding LSTM networks. Accessed, March, 31, 2021, http://colah.github.io/posts/2015-08-Understanding-LSTMs.

Ordóñez FJ, Roggen D (2016) Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors* 16(1):115.

Parmisano A, Garcia S, Erquiaga M (2020) A labeled dataset with malicious and benign IoT network traffic. *Stratosphere Laboratory: Praha, Czech Republic* .

Patterson J, Gibson A (2017) *Deep learning: A practitioner's approach* (O'Reilly Media, Inc., Sebastopol, CA).

Paxson V (1999) Bro: A system for detecting network intruders in real-time. *Computer Networks* 31(23-24):2435–2463.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-Learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

R Core Team (2020) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, URL http://www.R-project.org/.

Ryu S, Yang B, et al. (2018) A comparative study of machine learning algorithms and their ensembles for botnet detection. *Journal of Computer and Communications* 6(05):119.

Saad S, Briguglio W, Elmiligi H (2019) The curious case of machine learning in malware detection. *arXiv preprint arXiv:1905.07573* Accessed February 16, 2021, https://arxiv.org/pdf/1905.07573.pdf.

Segal M (2004) Machine learning benchmarks and random forest regression. *UCSF: Center for Bioinformatics and Molecular Biostatistics.* Accessed March, 22, 2021, https://escholarship.org/uc/item/35x3v9t4author.

Shukla L (2019) Designing your neural networks. *Towards Data Science.* Accessed February, 20 , 2021, https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed.

Stottner T (2019) Why data should be normalized before training a neural network. *Toward Data Science.* Accessed February 16, 2021, https://towardsdatascience.com/why-data-should-be-normalizedbefore-training-a-neural-network.

Van Rossum G, Drake Jr FL (1995) *Python tutorial* (Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands).

Wang P, Aslam B, Zou CC (2010) Peer-to-peer botnets. *Handbook of Information and Communication Security*, 335–350 (Springer, New York, NY).

Wang P, Sparks S, Zou CC (2008) An advanced hybrid peer-to-peer botnet. *IEEE Transactions on Dependable and Secure Computing* 7(2):113–127.

Woolf N (2016) DDoS attack that disrupted internet was largest of its kind in history, experts say. *The Guardian.* Accessed December, 10, 2020, https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.

Wrona K (2015) Securing the internet of things a military perspective. *2015 IEEE 2nd World Forum on Internet of Things*, 502–507 (IEEE).

Zebin T, Sperrin M, Peek N, Casson AJ (2018) Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks. *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1–4 (IEEE).

Zeek (2021) Zeek logs. Accessed June 16, 2021, https://docs.zeek.org/en/master/logs/conn.html.

Zheng DE, Carter WA (2015) *Leveraging the internet of things for a more efficient and effective military* (Rowman & Littlefield, Lanham, MD).

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California