NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

NETWORK TRAFFIC ANOMALY DETECTION ON A NAVY NETWORK

by

Michael J. Laws and Greg T. Bunder

June 2020

Thesis Advisor: Vinnie Monaco
Second Reader: John D. Fulp

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE NETWORK TRAFFIC ANOMALY DETECTION ON A NAVY NETWORK	**5. FUNDING NUMBERS**
6. AUTHOR(S) Michael J. Laws and Greg T. Bunder	
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) Fleet Cyber / 10th Fleet, Ft Meade, MD	**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)

Navy watchstanders are ill-equipped to monitor network status in real time, to include an inability to identify network anomalies and potential risks on-the-fly. This leads to a lack of situational awareness and ultimately an inability to determine the current network risk level. An existing unsupervised machine learning technique is identified and leveraged to enable the detection of anomalous DNS network traffic on a shore-based unclassified Navy network. The research conducted by the team outlines an architecture that could be extended to produce a capability to provide the watchstander a near real-time metric of a subset of the risk that the network is experiencing by classifying DNS traffic anomalies

14. SUBJECT TERMS big data, unsupervised machine learning, random cut forest, feature extraction, feature generation, INOSS framework, Amazon Web Services, GovCloud, AWS SageMaker, network architecture, network anomaly detection, unclassified Navy network, Hadoop, elastic map reduce	**15. NUMBER OF PAGES** 117
	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.

NETWORK TRAFFIC ANOMALY DETECTION ON A NAVY NETWORK

Michael J. Laws
Lieutenant, United States Navy
BS, U.S. Naval Academy, 2011

Greg T. Bunder
Lieutenant, United States Navy
BS, American Military University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Approved by: Vinnie Monaco
Advisor

John D. Fulp
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Navy watchstanders are ill-equipped to monitor network status in real time, to include an inability to identify network anomalies and potential risks on-the-fly. This leads to a lack of situational awareness and ultimately an inability to determine the current network risk level. An existing unsupervised machine learning technique is identified and leveraged to enable the detection of anomalous DNS network traffic on a shore-based unclassified Navy network. The research conducted by the team outlines an architecture that could be extended to produce a capability to provide the watchstander a near real-time metric of a subset of the risk that the network is experiencing by classifying DNS traffic anomalies.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**

1.1 Navy's Cyber Vision 1

1.2 FCC/C10F Tasking 1

1.3 Scope and Contributions 4

1.4 Thesis Outline 4

2 Background **7**

2.1 Machine Learning 7

2.2 Anomalous Network Traffic 11

2.3 Previous Research 15

3 Research Methodology **19**

3.1 Overview 19

3.2 Data Collection 21

3.3 Preprocessing. 21

3.4 Feature Selection 23

3.5 Machine Learning Algorithm 26

4 Code Implementation **29**

4.1 Overview 29

4.2 Data Capture 30

4.3 Feature Generation 30

4.4 Machine Learning 31

4.5 Inference 32

5 Results and Discussion **35**

5.1 Data Composition 35

5.2 Data Analysis and Visualization 37
5.3 Results Discussion 41

6 Implementation within INOSS Framework **45**
6.1 Overview 45
6.2 INOSS 45
6.3 AWS 47
6.4 Amazon SageMaker within INOSS 48

7 Conclusion and Future Research **51**
7.1 Overview 51
7.2 Thesis Limitations and Future Research 51
7.3 Key Takeaways 54

Appendix A Data Control Code **55**

Appendix B Data Capture Code **57**

Appendix C Data Transfer Code **59**

Appendix D Data Obfuscation Code **61**

Appendix E Spark Feature Pipeline Code **63**

Appendix F Machine Learning Code **75**

Appendix G Visualization Code **81**

List of References **95**

Initial Distribution List **99**

List of Figures

Figure 2.1 Depiction of Relationships of AI and Subsets. Source: [5]. 8

Figure 2.2 Machine Learning Models. Source: [6]. 9

Figure 2.3 OSI Model. Source: [8]. 12

Figure 2.4 IP and UDP Header. Adapted from [10]. 13

Figure 2.5 Confusion Matrix. Source: [12]. 16

Figure 3.1 NPS ERN 20

Figure 3.2 AWS Architecture. Source: [17]. 21

Figure 3.3 Head of CSV with 6-Tuple 22

Figure 3.4 Additional Features Generated 23

Figure 4.1 Data Flow Topology 29

Figure 4.2 Machine Learning Feature Generation 31

Figure 4.3 Machine Learning Code 32

Figure 4.4 Batch Transform Code 33

Figure 5.1 Normal Network Traffic 35

Figure 5.2 Graph of Network Activity over Time 36

Figure 5.3 Graphical Representation of the Linear Relationship between the Score and Feature 38

Figure 5.4 Graphical Representation of Feature over Time 39

Figure 5.5 Graphical Representation of Time and IP Source by Anomaly . . 40

Figure 5.6 Histogram of Anomalous IP Addresses 41

Figure 5.7 Standard Deviation Testing Results 42

Figure 6.1 INOSS Framework. Source: [26]. 46

Figure 6.2 Decision Analytics. Source: [27]. 47

Figure 6.3 Infrastructure Comparison 49

List of Acronyms and Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
C10F	Commander 10th Fleet
C2	Command & Control
CNO	Chief of Naval Operations
CONOPS	Concept of Operations
COS	Chief of Staff
CSV	Comma-separated Values
DAG	Directed Acyclic Graph
DDG	guided-missile destroyer
DEVOPS	Development Operations
DNS	Domain Name System
DODIN	Department of Defense Information Network
DODIN-N	Department of Defense Information Network - Navy
DoS	Denial of Service
EMR	Elastic MapReduce
ERN	Education and Research Network
FCC	Fleet Cyber Command

HCI Human Computer Interactions

HDFS Hadoop Distributed File System

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IEEE Institute of Electrical and Electronics Engineers

INOSS Integrated Navy Operations Support System

IP Internet Protocol

IPS Intrusion Prevention System

IRB Institutional Review Board

IT Information Technology

ITACS Information Technology and Communications Services

JVM Java Virtual Machine

ML Machine Learning

NEN Navy Enterprise Network

NETOPS Network Operations

NOC Network Operation Center

NPS Naval Postgraduate School

OSI Open Systems Interconnection

RRCF Robust Random Cut Forest

S3 Simple Storage Service

SA Situational Awareness

SAN Storage Area Network

SLA Service Level Agreement

SNMP Simple Network Management Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

UTC Universal Time Coordinated

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

First and foremost, we would like to thank Sarah Laws for her patience and editing assistance while we were working on this thesis. Thank you for supporting us through the long hours and frayed nerves as we worked through the thesis process.

We would like to thank Dr. John "Vinnie" Monaco of the Computer Science Department for helping us through this thesis. Thank you for your guidance and keeping us on track throughout the thesis process. Your assistance every step of the way was indispensable. Thank you.

We would also like to thank our friend, Dr. Eric Wayman, for lending his extensive mathematical knowledge and programming expertise. Without your help, this thesis would not have been possible. Thank you.

We would also like to thank the team at ITACS: CDR Chris Angelopolous, Kirk Benson, Craig Vershaw, Trent Hancock, Daryl Wilson, Ricky Estrada, CTNC Jackie Turner, Justin Brown, and other NOC and SOC personnel. This thesis depended on your expertise, flexibility, and infinite patience. Thank you.

Another thank you goes to Dr. Boger, CDR Borne, and the rest of the team that worked on the overall C10F project with us. The collaboration and accountability were invaluable.

Last, but most certainly not least, we want to give a big shout out to our fellow students in the CSO track, section 326-191CS. More colloquially referred to as "The Murder" (of the crow variety), JB, JS, Mitch, and Amanda, thanks for all the laughs, beers, and all around good times that kept us going during our time at NPS!

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Navy's Cyber Vision[1]

"Our Navy will protect America from attack, promote American prosperity, and preserve America's strategic influence. U.S. naval operations—from the seafloor to space, from the blue water to the littorals, and in the information domain—will deter aggression and enable resolution of crises on terms acceptable to the United States and our allies and partners" [1]. This is part of the mission statement that was promulgated by Admiral John M. Richardson, prior Chief of Naval Operations (CNO) in *A Design For Maintaining Maritime Superiority*. From this statement one can gather that now, more than ever, the information domain plays a vital role in protecting our country. Within the information domain falls the cyber domain and Fleet Cyber Command (FCC)/Commander 10th Fleet (C10F) have been charged with defending and delivering effects in and through cyberspace. From Fleet Cyber Command's (FCC) command page, part of their mission is to "conduct operations in and through cyberspace, the electromagnetic spectrum, and space to ensure Navy and Joint/Coalition freedom of action and decision superiority while denying the same to our adversaries" [2]. FCC and Commander 10th Fleet (C10F) are co-located in Fort Meade, MD, and also have a dual-hatted commander.

1.2 FCC/C10F Tasking[1]

In 2018, then FCC/C10F Chief of Staff (COS) Captain James Mills directed the N9 to head up research on visualization of data, Human Computer Interactions (HCI), and identify the most efficient way to display network health and status to a watchstander and decision-maker. Specific guidance included:

1. Using the Integrated Navy Operations Support System (INOSS) architectural framework, evaluate existing software tools for deployment to Department of Defense

[1]This section is a collaborative effort among multiple thesis students working on the same C10F funded project. The thoughts, ideas, and information contained within are not attributed to a single individual and can be found in the theses referenced in the subsections under Section 1.2.

Information Network - Navy (DODIN-N) watch floors, Network Operation Centers (NOCs), and for use by afloat Information Technology (IT) personnel.
2. Take into account industry and other government implementations, best practices, and employment of similar systems.
3. Integrate existing and novel malicious activity notifications into the INOSS framework to allow appropriate personnel the freedom to quarantine and investigate the activity.
4. Identify how these tools will support existing Navy programs of record.

This led to the funding for Naval Postgraduate School (NPS) research to be conducted by thesis students. Dr. Dan Boger was appointed as the project manager; Dr. Boger then selected students with skill sets that aligned with the requirements of the aforementioned research. The students, their thesis topics, and their respective deliverables are described in the following subsections.

1.2.1 Architecting Autonomous Actions in Navy Enterprise Networks

Dr. Dan Boger and Dr. Luqi are co-thesis advisors for Lieutenant Max Geiszler. This thesis investigates Navy Enterprise Networks (NENs) in an attempt to better understand the fundamental operation of the Navy's networks. The main idea behind the research is to explain how NENs can conduct Network Operations (NETOPS) to meet unique Navy mission sets and ensure adequate information is given to higher up organizations. The investigation covers some of the use-cases in which the Navy has an intensive need for human-driven processes to accomplish necessary critical tasks. It also explores where man-hours are being inefficiently spent due to process redundancy and limited human watch-stander proficiency. It then suggests a technical architectural change to NEN infrastructure utilizing the INOSS framework which helps to facilitate automated solutions to problems which have been presented by FCC/C10F, and also suggests a change to tightly integrate Development Operations (DEVOPS) in operational processes.

1.2.2 Network Traffic Anomaly Detection on a Navy Network

Dr. John Monaco is the thesis advisor for Lieutenant Mike Laws and Lieutenant Greg Bunder. This thesis determines the viability of using existing unsupervised machine learning techniques to detect anomalous network traffic from an unclassified Navy network. Upon completion, this thesis gives a recommendation as to whether unsupervised machine

learning can be used for anomaly detection. Detailed analysis of implemented features that are most effective for anomaly detection, along with any lessons learned and obstacles met during research, are provided. Lastly, this thesis addresses what an architecture might look like that would be used to implement network anomaly detection via unsupervised Machine Learning (ML) within the INOSS framework.

1.2.3 Visualization: Functional and Conceptual Approach to Network Operations

Dr. Dan Boger is the thesis advisor for Commander Henry Lee Bush. The thesis analyzes the current visualization at various organizations, the private sector, and the public sector, to better understand how visualization provides a network's health and status. The main idea behind the thesis is to evaluate visualization in key focus areas: single pane of glass, information immersion, information framework, and information concept. It does this by covering case studies that were done through the site observation to identify how information is collected, processed, analyzed, and visualized to support command and control of the network. Through the case studies, the thesis also reviews the information not captured because of stovepiped systems, limited shared management information, and manual process which reduces the information in visualization. The information not captured in turn impacts situation awareness and decision making which negatively impacts command and control of the network. The thesis recommends the use of the INOSS functional framework to improve processes to support visualization of information and information immersion through space design. Lastly, it introduces an information management concept to support command and control of the network.

1.2.4 How Information Sharing Affects Network Operations

Dr. Dan Boger is the thesis advisor for Lieutenant Eva Castillo. Effective information sharing between various components are crucial to FCC/C10F successfully and efficiently meeting the mission. The thesis has two goals. The primary goal is to examine whether existing information systems, mandates, policies, or Service Level Agreements (SLAs) are limiting information sharing within the FCC/C10F organization. The secondary goal is to seek technical and non-technical solutions to support current and evolving requirements. The thesis will evaluate solutions studied that can positively impact information

sharing for the organization. Research approaches include interviews and observations in academia, civilian IT sector, defense organizations, programs of record, and Tier 1, 2 and 3 providers. From the research gathered, conclusions were drawn to the effectiveness of current technologies, mandates, and policies along with proposed solutions.

1.3 Scope and Contributions

The Navy currently uses Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) to protect its networks. These IDSs and IPSs use human-enabled rules to filter, detect, and prevent anomalous activity. The rules are only as good as the person writing them and their knowledge of the network. Similar limitations exist for supervised machine learning as it also requires human labeling. An unsupervised model, when properly created, has the potential to not require human intervention to accomplish the same results.

The Navy also uses a suite of disparate and aging tools to gain situational awareness of events that occur within the network boundaries. The average watchstander tasked with maintaining access to these tools often does not possess the requisite ability or time to collate the information that is contained within. The inability to process, decide, and act has serious ramifications within the commanders' decision cycle.

This thesis will examine the viability of detecting anomalous unclassified Domain Name System (DNS) traffic from an individual Navy network. Depending on the role of the command that uses the network, traffic can look very different. For example, traffic from an Arleigh Burke Destroyer can look very different from traffic from a shore installation or the NPS network, but given any Destroyer, one could suppose that traffic would be relatively similar. This is due to many variables such as the equipment on board, number of personnel, and mission set. During this process, we will also determine an extensible architecture that can be integrated within the INOSS framework. Specific excluded areas include classified networks and large ashore or afloat networks.

1.4 Thesis Outline

Chapter 2 provides background context for the reader to better understand the methodology and results that are presented in later chapters. It includes an overview of what Artificial Intelligence (AI) and ML are, the different types of ML, a detailed description of network

anomalies, and how ML and network anomalies relate. Background is given on some related research that has previously been conducted using ML to detect network anomalies.

Chapter 3 outlines the specific methodology that was used to conduct the research. It explains the process from the collection of network data through an overview of feature generation.

Chapter 4 elaborates on the code utilized to extract features, run the ML algorithm, and ultimately score anomalies found within the network traffic.

Chapter 5 provides a detailed analysis of the results obtained from the ML algorithm.

Chapter 6 begins with a brief description of the current architecture that Navy networks fall under. This is followed by a description of how the methods employed within the thesis could be used to fit into the existing Navy architecture.

Chapter 7 concludes the thesis. Along with the conclusion, lessons learned and opportunities for future work are provided.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

2.1 Machine Learning

Exactly when the first example of ML was introduced, and by whom, is somewhat ambiguous, but the concept has been around since at least as early as the late 1940s. In a lecture given to the London Mathematical Society in February 1947, it was postulated by none other than Alan Turing, regarded by many as a founding father of AI, that "what we want is a machine that can learn from experience" [3]. Turing's work laid the groundwork for AI, and by proxy ML, as we know it today.

So, what is ML and how exactly does it relate to AI? AI is loosely defined as "the branch of computer science that is concerned with the automation of intelligent behavior" [4]. Intelligent behavior, in this case, would be behavior that mimics a human being's ability to learn and apply concepts and skills. Contained within the overarching field of AI is ML, which like AI, is the automation of intelligent behavior. However, what sets ML apart is that it is not explicitly programmed to do so; hence the term "learning." Rather than predefined parameters to provide intelligence, the intelligence is instead learned. Intelligent behavior can be mimicked by a savvy programmer willing to put in the time and effort but ML is distinct in its ability to learn from inferences and not solely from the underlying programming. Also contained within AI, but not relevant for the purposes of understanding the work presented in this thesis, is deep learning, a further subset of ML. Figure 2.1 depicts a visual Venn diagram representation of the relationships between AI, ML, and deep learning.

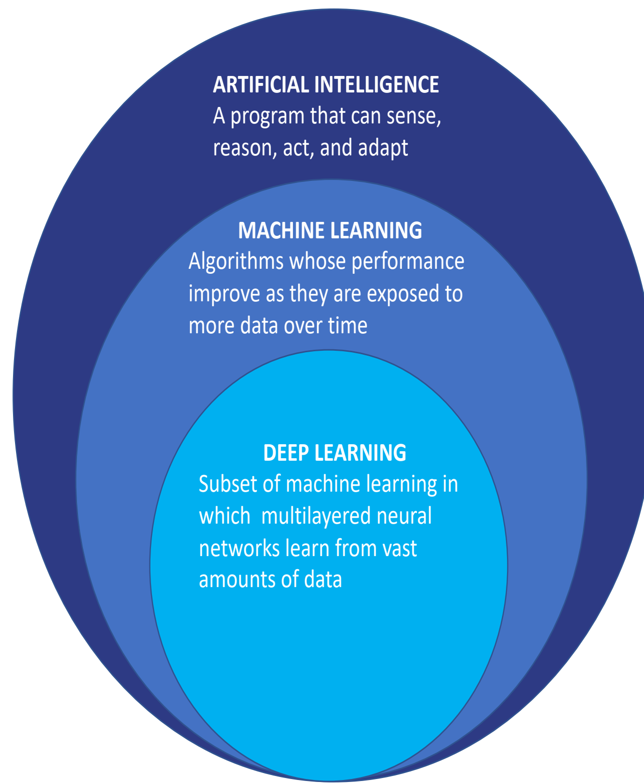

Figure 2.1. Depiction of Relationships of AI and Subsets. Source: [5].

ML can be further separated by the type of learning algorithms involved. The three main categories are supervised, semi-supervised, and unsupervised. Figure 2.2 shows the relationship between the three main categories of algorithms and how the outputted models differ based on the type and composition of the training data that the algorithm is applied to. For example, an unsupervised ML algorithm would be run on all unlabeled data and this would generate an unsupervised ML model.

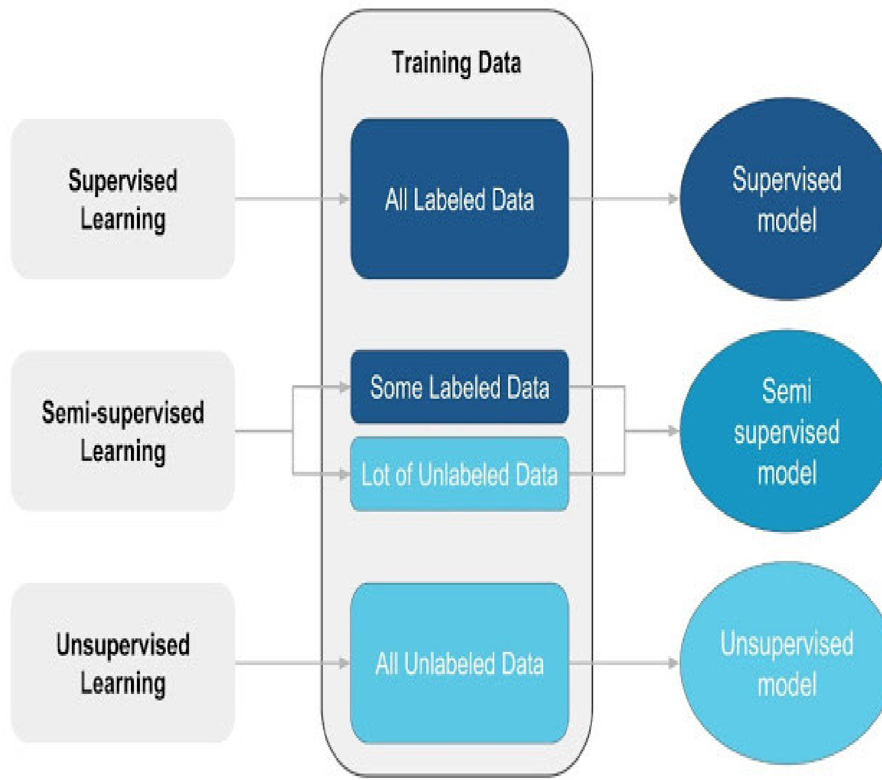

Figure 2.2. Machine Learning Models. Source: [6].

A dataset within ML is broken up into two or three groups depending on the programmer. The first group of data will always be the training dataset. The general rule of thumb is to use seventy percent of the data to train with and is what the model will learn on. The other thirty percent can be used as one set of data for validation and testing or it can be further divvied up into twenty percent for validation and ten percent of testing data. Validation occurs after the initial training and is used to fine-tune the model. The test set is applied last and is used to determine the effectiveness and performance of the model. The ML pipeline can then be broken into phases that correspond to the dataset groups.

The input into a ML model are features. These can be generated by a human or, in the case of deep learning, by the machine. Features are characteristics or traits of the dataset that can be measured in some way. For example, if one were to have a dataset of a network capture that included characteristics such as packet length, destination port, source port, etc., then each of those traits could be used as a feature. In most cases, not all of the features of a

dataset will be relevant for the learning involved, and therefore feature generation will be conducted. This can include removing unnecessary features and transforming others. The output of a ML algorithm is a model that can make predictions on new data.

2.1.1 Supervised

Supervised ML builds a model based on mapping an input to an output. All the data that is used for training in supervised models are labeled, hence why it is referred to as supervised. The two most commonly used techniques for supervised learning are statistical classification and regression analysis. Statistical classification takes an input and then maps it to a category as an output. An example of this would be taking input of various features of a flower and then the model would output the type of flower. Because the output is assigned a category and not a value, it is statistical classification. Regression analysis, on the other hand, uses the input and provides a continuous value as an output. An example of a regression algorithm would be to take input such as salary, job title, and the type of car a person drives and then based on that, provide an output that attempts to predict the age of a person. In the regression model, there are no specific categories and in this case, it is a numerical sliding scale.

2.1.2 Semi-supervised

Semi-supervised ML uses a combination of labeled and unlabeled data within its training dataset. A popular algorithm within semi-supervised ML is self-learning. This algorithm uses the data that has already been labeled by a human to try and label the rest of the unlabeled data. It will go through many iterations of this due to the fact that it will only label the unlabeled data when it has a high confidence of the correct prediction. This type of ML model can be useful for when there is so much data that it is either unfeasible or inefficient for a human to go through and label everything within the dataset. A human would then go in and label a percentage of the data and allow the algorithm to do the rest of the work. An example of this would be if a ML application was trying to detect fraud. Using supervised ML and labeling all of the fraud in the dataset might not be feasible since there may be unknown examples of fraud in the dataset. In this case, we would label the data we could and then have the ML algorithm attempt to label the rest of the data.

2.1.3 Unsupervised

Unsupervised ML is different from the previous two categories due to the fact that it contains no labeled data. All data within the training dataset is unlabeled. Because there is no labeled data, unsupervised ML models primarily use one of two methods to analyze the data. The first method is called clustering and algorithms that use this method will group data together by commonalities. Algorithms that use cluster analysis are ideal for situations where the output desired is either data that is of the same kind, or when searching for anomalies. Anomaly detection is a special case of clustering where there are two clusters: normal and anomalous. Another method that may be used is visualization and dimensionality reduction, which consists of using the model to visually represent the data as either two-dimensional or three-dimensional. In this format, a human can discern patterns within the data, as well as potentially see gaps within the patterns or outliers, such as anomalies. Generative models, ML models that output how likely something is to occur, could also be considered unsupervised. Unsupervised ML tends to be the most applicable to real-world applications because most data is not labeled. For instance, many eCommerce and advertisement ML applications use unsupervised learning.

2.2 Anomalous Network Traffic

In order to understand anomalous network traffic, one must first understand what constitutes normal network traffic. It is also worth noting that just because traffic is anomalous, it does not necessarily mean it is malicious. Network traffic consists of data that flows across defined architectural layers between networks. Used for the purposes of this thesis is the Open Systems Interconnection (OSI) model. The OSI model was proposed in 1977 and was presented as a discussion on a model that allowed all computers to communicate with one another with a common architecture [7]. Figure 2.3 depicts the layers of the OSI model.

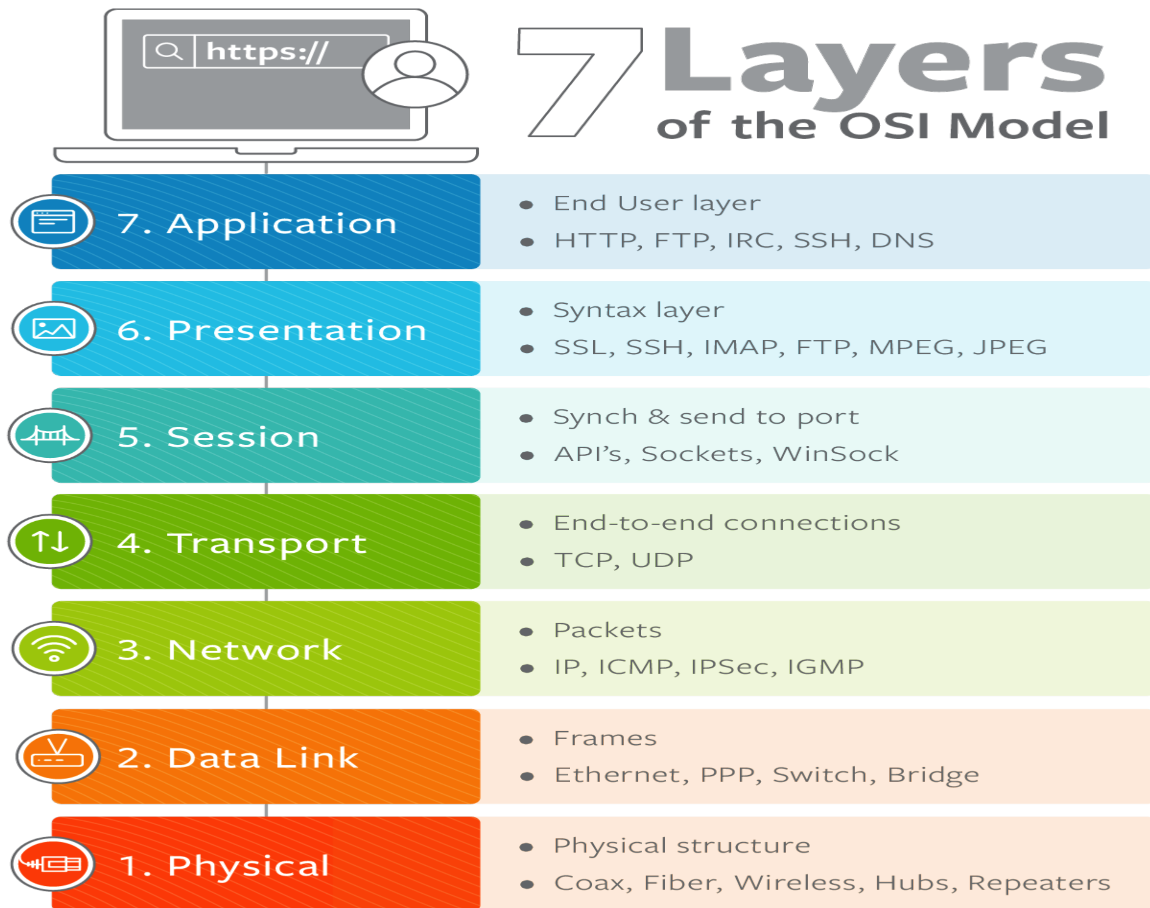

Figure 2.3. OSI Model. Source: [8].

A network is comprised of two or more computer systems that interact with one another and can communicate back and forth. It can be connected via a physical medium, such as Ethernet, or wirelessly, which is the case with the Institute of Electrical and Electronics Engineers (IEEE) 802.11 protocol, commonly referred to as WiFi [9]. Devices on a network are constantly communicating through the use of protocols. There are many common protocols, to include Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), and Internet Protocol (IP) to name a few. Protocols tend to be for a specific purpose. For example, ICMP is used for email, TCP is used to establish and maintain a network communication, and UDP is also used for communication but does not establish a connection or make sure that the hosts receive all

of the packets sent like TCP does. One of the most common protocols, UDP, is leveraged in this research.

The OSI Model layer 3 contains the network data in the form of packets. These packets can further be broken down into headers. Headers contain protocol specific information that can be extracted into features that can be processed by a ML algorithm. The initial features that were extracted from the IP and UDP header consisted of the source and destination IP addresses, source and destination ports, and UDP packet length. UDP packet length includes the combined length of data and the UDP header. This is sometimes referred to as a 5-tuple and the components of the tuple can be seen in Figure 2.4.

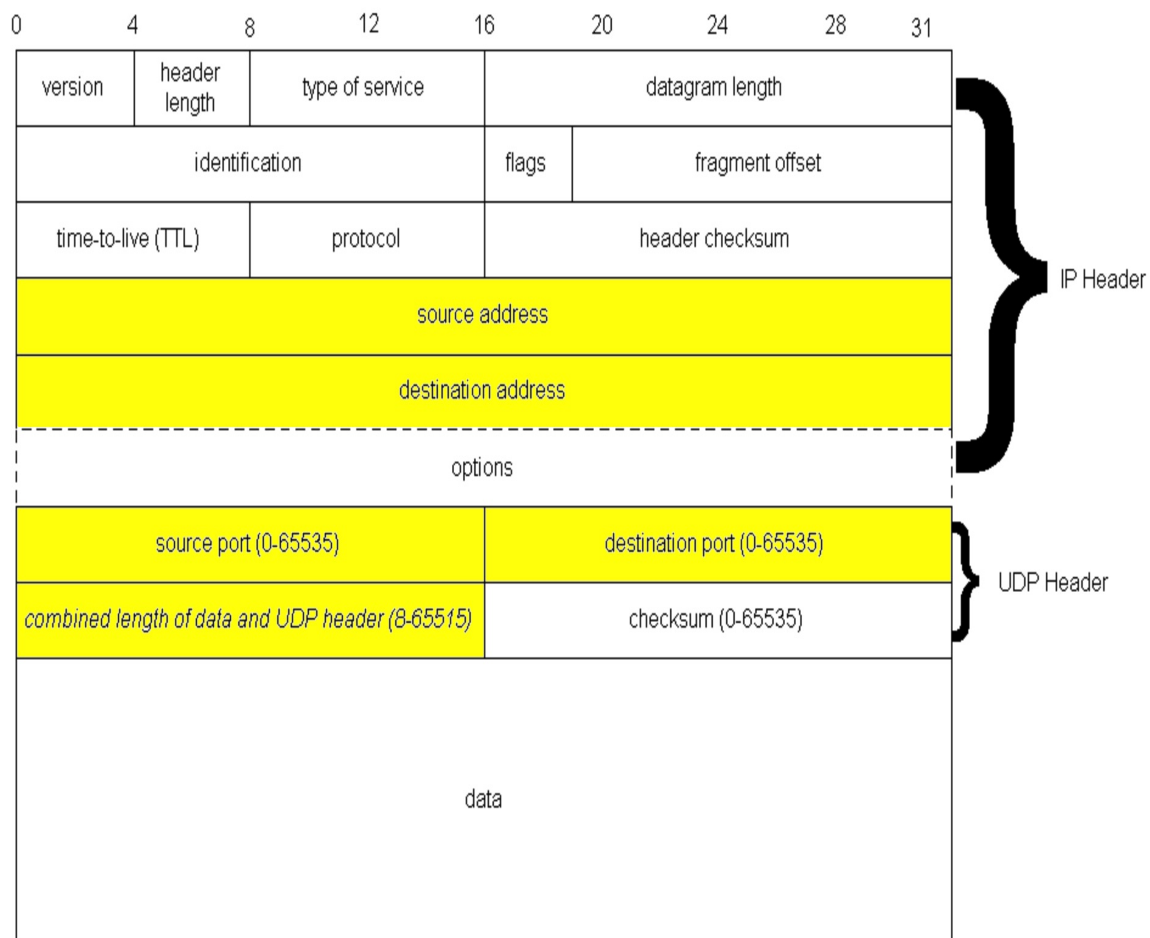

Figure 2.4. IP and UDP Header. Adapted from [10].

Different networks have different norms, therefore anomaly detection tends to be network-specific. For instance, a guided-missile destroyer (DDG) unclassified network would see different traffic than a shore based Navy unclassified network, like at NPS. The two networks would have different purposes and therefore much of the data seen transiting would vary in volume and shape. For example, the circadian rhythm of a ship at sea would be essentially a 24/7 flow of data, whereas the NPS unclassified network's circadian rhythm would tend to follow a traditional work week cycle, to include peak hours during 9-5 and lulls overnight and on weekends. Another difference between the two aforementioned networks could be the types of devices that are sending data across. For example, a ship would have systems that help to identify friendly units and generate a common operating picture for situational awareness that a shore-based network like NPS does not have. The following are some examples of abnormal traffic that could be seen across a variety of different networks.

2.2.1 Misuse of a Protocol

One type of anomaly that may exist is when a specific protocol is used outside of its intended purpose. An example of this is the use of the DNS protocol. The DNS protocol is used to translate addresses from something human-readable to the underlying IP address that belongs to the website or server a person is trying to reach. How the protocol works in the simplest terms is the user will send a DNS query and the assigned DNS server will return a response. How a malicious user could misuse this protocol is to create a covert channel using any number of techniques. A covert channel is a communication channel that occurs in a way that is not transparent to the network owner. An example of a very rudimentary way of conducting DNS covert channel communications would be for the malicious user to have their own DNS server that they would query. Then, from the network they are on, they would have an odd number of queries equal a binary zero and an even number equal to a binary one. Once received, the ones and zeroes would be put together into strings of binary for whatever purpose chosen by the malicious user [11]. This is something that is not immediately noticeable to a network administrator due to the fact that DNS queries and responses are completely legitimate traffic. The anomaly in this situation would be that there would be an extremely high number of queries and responses between the malicious user and their DNS server.

2.2.2 Excessive Latency

Another anomaly that often presents itself is excessive latency. Latency is a norm in networks, as there is always some sort of time delay inherent in network communications. When it becomes an anomaly is when the delay is greater than what is considered by the network administrators as normal. This is one anomaly that is not necessarily malicious and could be tied to something as simple as hardware or routing issues within a network. However, it could also be indicative of something as malicious as a man-in-the-middle attack. In a man-in-the-middle attack, a user's network traffic is routed through a malicious user unbeknownst to the original user. It is also possible in these cases for the malicious user to alter the traffic that is being communicated through them.

2.2.3 High Volume of Traffic

While a high volume of traffic from a specific source or to a specific destination address is not necessarily malicious, depending on the volume it could indicate an issue. Misconfigurations and hardware or software errors could cause a heavy volume of traffic. Malicious traffic, most commonly in the form of Denial of Service (DoS) attacks, could also cause this type of anomaly. A DoS attack is an attack on a network that is meant to overwhelm a node's resources and result in a degraded or down node. This is done either through such a large volume of data that it overwhelms some component of the network or in some cases, a malformed packet that could crash a component of the network.

2.3 Previous Research

To date, there has been a significant amount of research done in the area of anomaly detection using ML. Most of the research has largely revolved around the use of supervised or semi-supervised ML, however, there has also been research conducted with unsupervised ML. This area of research is relatively new, with the majority of the research occurring within the last decade. The accuracy of the different techniques varies as does their ability to detect false positives and false negatives. A true positive and true negative are when the predicted values match the actual values. In the case of anomaly detection, this would mean that when the ML model detects an anomaly, it is actually an anomaly. A false positive and false negative are when the predicted values do not line up with the actual values. A false positive would be if the model flagged something as anomalous when in actuality it was not

an anomaly. A false negative would be if something was an anomaly and the ML model did not identify it as such. Figure 2.5 shows a true/false and positive/negative confusion matrix.

Predicted Value (predicted by the test)		Actual Value (as confirmed by experiment)	
		positives	negatives
	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

Figure 2.5. Confusion Matrix. Source: [12].

2.3.1 Supervised Machine Learning Application

In 2018, at an Intelligent Systems Conference, a joint research paper was produced by students from Princess Sumaya University for Technology and Mutah University. In this paper, the students laid out how they used three different supervised ML techniques for the purpose of detecting DoS attacks. They built ML models using Random Forest, Ada Boost, and multilayer perceptron classifiers. These classifiers were used to take in Simple Network Management Protocol (SNMP) data that included information from the network routers, switches, and hubs. The result of their research was that the Random Forest classifier performed the best and was able to detect anomalies with a 99.93 percent accuracy and the other two classifiers were not far behind. Ada Boost MI detected anomalies with 99.60 percent accuracy and MLP with 98.86 percent accuracy [13].

2.3.2 Unsupervised Machine Learning Application

In 2007, at the 2nd USENIX workshop on tackling computer systems problems with ML techniques, a research paper was published by three students from McGill University in Montreal, Canada. This paper presented how unsupervised ML could be used to detect

network anomalies, such as attacks on the network or high volumes of traffic. The students used two different ML techniques: One-Class Neighbor Machine and Kernel-based Online Anomaly Detection. The ML techniques were each applied to two different datasets. One of the datasets was the Abilene network, which includes 11 core routers. The other dataset was a set of still image captures from a traffic webcam in Quebec. Although not a network-related dataset, this was used as evidence of the viability of using unsupervised ML for anomaly detection. The results of this research were not quantified; however, the researchers conclude that both unsupervised ML techniques applied were successful but needed tweaks to be more efficient. Of the necessary changes identified for future work two of them that stood out were the need for the algorithms to run in real-time and to adapt as the incoming data changes [14]. The latter of the previous challenges is typically referred to as distribution shift, which means that what we consider normal and anomalous behavior will change over time.

2.3.3 Robust Random Cut Forest

In 2016 a paper was written that delved into the use of the Robust Random Cut Forest (RRCF) algorithm to detect anomalies, specifically anomalies within a data stream. A data stream is a unique problem set for anomaly detection due to the fact that it is not limited to solely a snapshot. In previous research that involved ML for anomaly detection, the research was conducted against a static set of data. This sort of application works when there is a finite amount of data, but for the real-world application of detecting anomalies within a network, one would not want to be limited to just one portion of data. The preferred method would be to input a stream of data and conduct anomaly detection as the data flows through the architecture. The research team discovered that regardless of the sample size used RRCF was more accurate at detecting anomalies (in some cases the accuracy over doubled) than the baseline Isolation Forest algorithm. In addition to a significant increase in accuracy, RRCF was also less likely to flag false positives [15].

A team from the University of Michigan wrote another research paper that further built on the work of Guha et al. in 2019. This paper produced the first open-source implementation of RRCF for other researchers to build on and experiment with. With their open-source code Bartos et al. conducted two tests to validate the work previously conducted in the 2016 study. The first test confirmed that RRCF did, in fact, detect outliers that One-Class

SVM and Isolation Forest algorithms were unable to. The second test was conducted to validate that RRCF was better at detecting the onset of an anomaly than the Isolation Forest algorithm [16].

CHAPTER 3:
Research Methodology

3.1 Overview

The network data used for research was collected from the NPS Education and Research Network (ERN), reviewed and approved by the NPS Institutional Review Board (IRB), then released by the Information Technology and Communications Services (ITACS). In order to test the effectiveness of the algorithm and ML model implemented, network traffic was collected from two separate locations, as shown in Figure 3.1. The first location was outside of the NPS firewall where traffic leaving the network was already filtered but traffic coming in was not. The second location was inside the firewall where traffic was filtered coming in but was not filtered leaving the network. After the network data was collected, it was then cleaned up to include only features relevant for our research and turned into a format that could be uploaded into an Amazon Web Services (AWS) Simple Storage Service (S3) bucket for ingestion into Amazon SageMaker, the ML service, and Application Programming Interface (API) used to train, test, and deploy the ML model. An API is an interface between a program and a library, which adds additional functionality to the program. The AWS infrastructure is shown in Figure 3.2. Once the data was uploaded to AWS S3, additional features were generated.

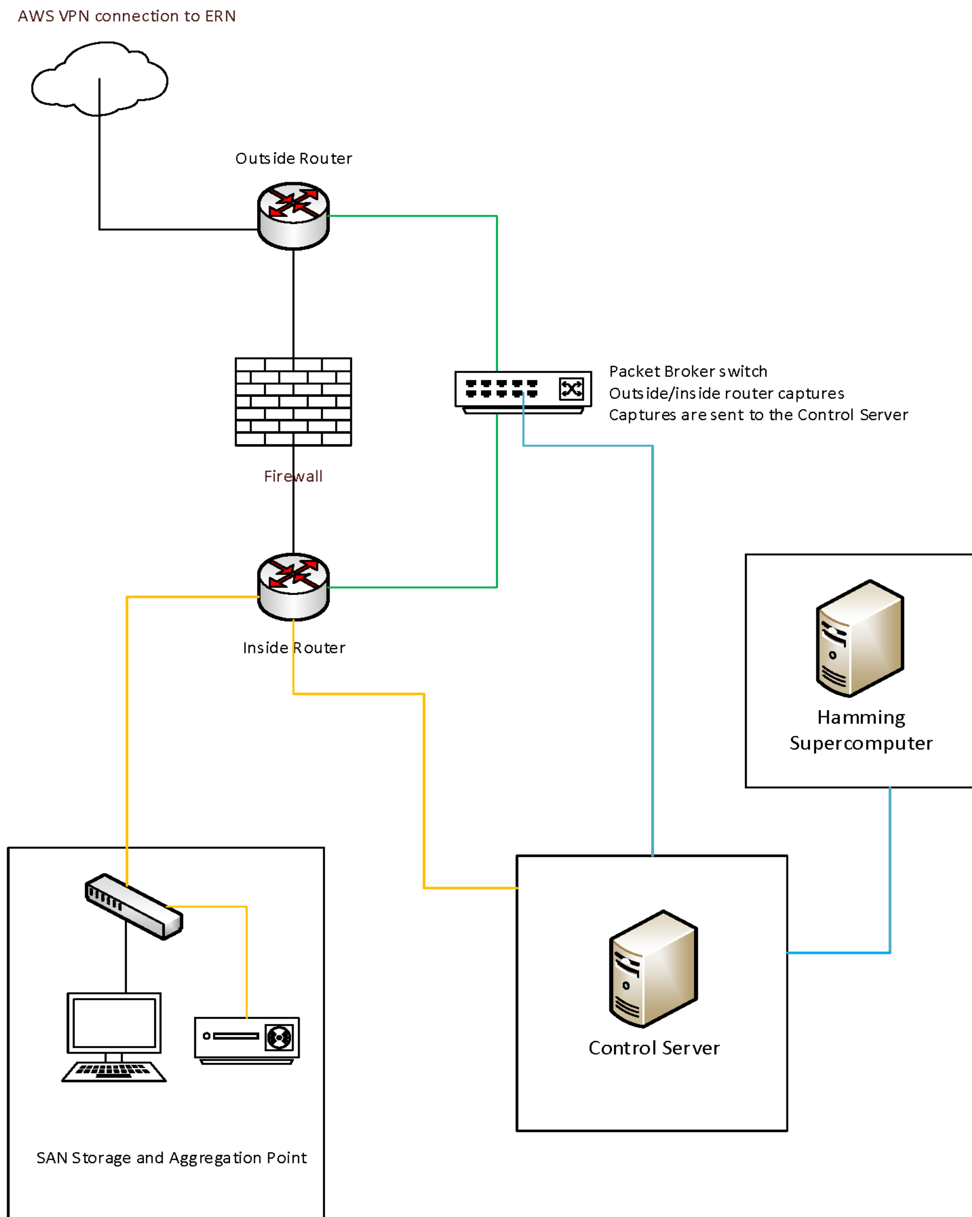

Figure 3.1. NPS ERN

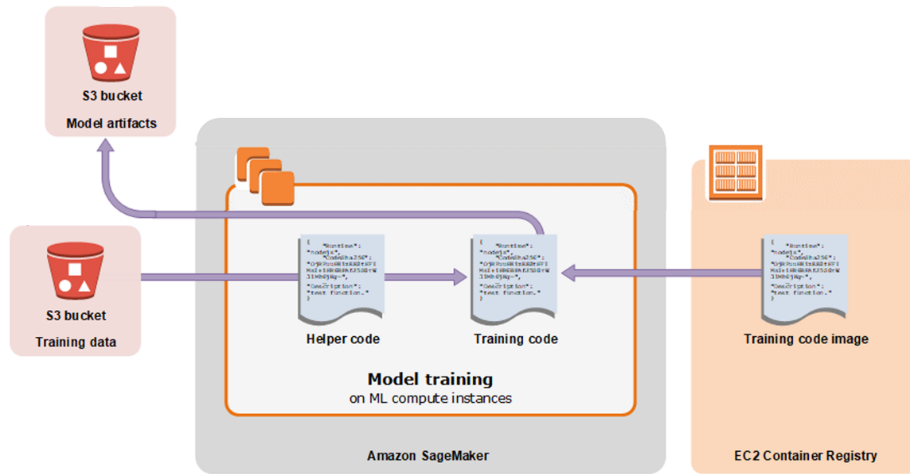

Figure 3.2. AWS Architecture. Source: [17].

3.2 Data Collection

Shown in Figure 3.1 is the flow of data as it traverses the NPS ERN from capture point to AWS. The first capture point is at Outside Router and is outside of the firewall. The data coming into the network at this capture point is not filtered by the ERN firewall but traffic that is leaving the network is. The second capture point is at Inside Router, which provides data that is unfiltered when leaving the network but incoming traffic that is filtered by the ERN firewall. The data from both of these capture points is then sent through the Packet Broker switch to the the Control Server. The Control Server then pushes the files through a dedicated 10 Gb/s connection to Hamming, NPS's supercomputer. The file format for capture is PCAP, a common format for network traffic captures. Scripts located on the Control Server automate this entire process. See Appendix A, Appendix B, and Appendix C for associated code.

3.3 Preprocessing

The format that was ingested into the ML model is Comma-separated Values (CSV). CSV files separate the different values by comma, offer a tabular format, and are commonly read by programs and APIs, such as Scala, Apache Spark, and Python. Converting to CSV linearly, one-by-one, is incredibly time consuming, and in order to convert pcap files to

CSV in parallel, a large amount of processing power was necessary. It is for this reason that Hamming was used for the conversion. During the conversion to CSV, the pcap files were stripped of data irrelevant to the research. The conversion was automated by a script that used TShark, a command-line interface that allows for the same network packet analysis as Wireshark.

The data that was retained from the pcap files during conversion to CSV files was filtered to only include the DNS protocol and consisted of the 5-tuple plus epoch time, making it a 6-tuple. This can be seen in Figure 3.3. Epoch time is a standardized time that all computers worldwide use and is the total elapsed time in seconds from January 1, 1970 at 00:00:00 Universal Time Coordinated (UTC). Epoch time for this capture was the time the packet was seen by the Control Server.

frame_time_epoch	ip_src	ip_dst	udp_len	udp_srcport	udp_dstport
1.581148801755486E9	[redacted].97.179	[redacted].20.12	42	52406	53
1.581148801768462E9	[redacted].22.154	202.12.27.33	36	38408	53
1.581148801769249E9	[redacted].56.50	[redacted].20.12	43	52090	53
1.581148801769254E9	[redacted].56.50	[redacted].20.12	43	52090	53
1.581148801769695E9	[redacted].107.147	[redacted].20.11	39	53985	53

Figure 3.3. Head of CSV with 6-Tuple

Upon completion of preprocessing the CSV files were uploaded to an AWS S3 bucket for additional feature generation prior to training a ML model, as shown in Figure 3.4. The connection from the NPS ERN is highlighted in Figure 3.1.

	ip_src	time_window	Sum_Packet_Length	Packet_Count	average_packet_len
0	1.710655e+08	2635274	48422	1036	46.739382
1	2.887019e+09	2635274	9111	133	68.503759
2	2.902262e+09	2635274	6096	104	58.615385
3	2.887015e+09	2635274	7838	108	72.574074
4	2.887019e+09	2635274	508	10	50.800000

Figure 3.4. Additional Features Generated

3.4 Feature Selection

After the CSV files were uploaded to an AWS S3 bucket, the data was streamed into a series of services and frameworks to generate features and prepare the data for ingestion into the ML model. The following programming languages and frameworks were used to accomplish this.

- Python — Python is a high-level programming language that is very human-readable and due to this, easy to code with. However, it is very inefficient with its memory usage and is unable to conduct parallel processing. Additionally, the basic scripts for data transfer were written in Python and the Amazon SageMaker API can be accessed through Python.
- Scala — "Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its Java Virtual Machine (JVM) and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries" [18]. Scala was used to generate features by combining columns over time periods to create relationships within the data that added additional insight into the data that was not originally there. Scala can handle large volumes of data and conduct parallel processing very quickly due to its use of Spark as a framework.
- Spark — Spark is a "unified analytics engine for large-scale data processing. It achieves high performance for both batch and streaming data, using a state-of-the-art Directed Acyclic Graph (DAG) scheduler, a query optimizer, and a physical

execution engine." It can be used within a variety of other programming languages, such as Python, Scala, and Java, and is ideal due to its ability to efficiently handle large volumes of data. It can also be run on existing frameworks, in the cloud, or by itself as a standalone [19].

- Hadoop MapReduce — "Hadoop MapReduce is a software framework for distributed processing of large datasets on compute clusters of commodity hardware. It is a sub-project of the Apache Hadoop project. The file system, Hadoop Distributed File System (HDFS), is a distributed file system designed to run on commodity hardware, is fault-tolerant, and provides high throughput that is beneficial for applications that have a large data set [20]. The framework takes care of scheduling tasks, monitoring them and re-executing any failed tasks" [21].

The first five features that were chosen were the 5-tuple. On their own, without viewing their relationship with each other, each of these can be individually observed by the ML algorithm for anomalous behavior, however not very effectively. They were all chosen because they relate to DNS traffic and are necessary to detect inter-relational anomalies within the dataset. The following are the 5 basic features that were looked at and what anomalous behavior of those features might represent when inter-related with other features. The 5-tuple features were looked at as they related to Epoch time, and the inclusion of Epoch time makes it a 6-tuple.

- Source IP Address — The source IP address is the IP address of the initial sender of a packet. An anomaly that could be detected with this is a high volume of DNS packets from a unique source IP address over a time window.

- Destination IP Address — The destination IP address is the IP address of the initial sender of a packet. Using the destination IP address, the same type of anomalies as with using the source IP address can be detected. This is useful when looking at traffic that is inbound to the network.

- Source Port — The source port is the port number that a packet originated from. If the sender is a client then the port number will most often be greater than 1023, whereas if it is a server, it will be a common port number. Common port numbers are generally tied to a specific protocol, for example, port 53 is typically DNS traffic. If

traffic is identified that is DNS and it is not over port 53, and is also not greater than 1023, this could potentially be anomalous.

- Destination Port — The destination port is the port number that a packet is being sent to. If the packet originated from a client then the destination port will most often be a common port number, whereas a server's destination port will be a port number greater than 1023. Anomalous behavior in this situation would be similar to that of a source port.
- UDP Packet Length — The UDP packet length is the total length of a package in bytes. For DNS traffic, the length may vary but it would be anomalous if the average packet size was 256 bytes for instance and a packet came through that was 512 bytes.

The features just discussed were simple in terms of their relationships. However, more complex features can be generated by writing code that develops relationships between two or more CSV columns and includes this in a separate column in a new CSV. One of the most important relationships is that of the features as seen over a sliding time window. The length of the time window will add or remove granularity. If the time window is too narrowly scoped then it could miss anomalies that take place over a large time window and conversely if the time window is too broadly scoped it may miss anomalies that appear normal due to repetitive occurrences over a larger time window. These features can be further grouped into the categories of descriptive, histogram, and contextual. The following features were used in this thesis:

3.4.1 Descriptive Features

Descriptive based features take in multiple values and use statistical methods to compress them down to a single scalar value. Descriptive based features that were used were:

- DNS Traffic Volume — This feature created a relationship between source and destination IP address and number of packets over a rolling time window. Any source IP addresses that were identified as an NPS DNS server were dropped from the dataset as the feature was created. This feature identified unique source IP addresses that were sending or receiving high volumes of DNS traffic to a unique destination IP address. Traffic originating from NPS DNS servers were dropped from the dataset because it

is a known that the majority of the packets should be to or from the school's DNS servers.

- Packet Length Distribution — This feature created a relationship between source or destination IP address and UDP packet length. Unique IP addresses were looked at over a window of time and their total UDP packet lengths were averaged and then compared against all the other unique IP addresses. Potentially anomalous behavior that was identified were the outliers that fell outside of the overall average packet length. NPS DNS servers were also dropped from this dataset.

3.4.2 Histogram Features

A histogram is a visual representation via bars on a graph of the distribution of items over an axis. This axis could be time, a count of something, or another unique identifier. The bars that are depicted by the histogram are a representation of a range of values, referred to as bins. The use of bins allows results to be grouped together and a cleaner visualization of the resulting outputs [22]. Histogram features can be either normalized or contain frequency counts. The histograms in this thesis use frequency counts, meaning that counts are provided for each bin, and are not normalized to scale values relative to one another. The following is the histogram-based feature that was used:

- Frequency Distribution of Packet Sizes — This feature takes unique source IPs and calculates the frequency of packet size ranges over a time window. Anomalous behavior that would be beneficial to identify is source IP addresses that is sending a large number of overly large packets over the time window. The packet size ranges are predefined and referred to as bins. Bin sizes were determined through an implementation of the Freedman–Diaconis rule, which uses the Interquartile Range, to group ranges into bins. The Interquartile Range takes a range of values, divides it into four equal parts, where each of the dividing lines is a quartile, and the output is the third quartile value minus the first quartile value [23].

3.5 Machine Learning Algorithm

The Amazon SageMaker implementation of RRCF was chosen because it has several beneficial characteristics that make it the best fit for this problem set. As described in

Chapter 2, among other things RRCF excels at anomaly detection in streaming data. It is also resilient to irrelevant data, so if one of the features that is provided does not produce value, the algorithm will inherently devalue its presence. The algorithm provides what is known as scoring, which is an essential element to determining if a data point is considered anomalous or not. The scoring varies by application and is not necessarily just a percentage. The lower the score, the less likely that a data point is an anomaly, and conversely the higher the score, the more likely it is that a data point is an anomaly [24]. The cutoff for what is considered anomalous can be set by the user and should be set to produce the fewest number of false negatives possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Code Implementation

4.1 Overview

The purpose of delving into the code is to describe how the features enumerated in the previous chapter were created and utilized to translate large quantities of information into a small amount of useful information for the network operator. This information flow is illustrated in Figure 4.1. Source code documents are divided by programming language and functionality and may be found in the appendices. NPS specific information was scrubbed from the code, including items such as network bits in IP addresses, DNS server information, and AWS S3 bucket information. All scrubbed information is annotated in the code with comments.

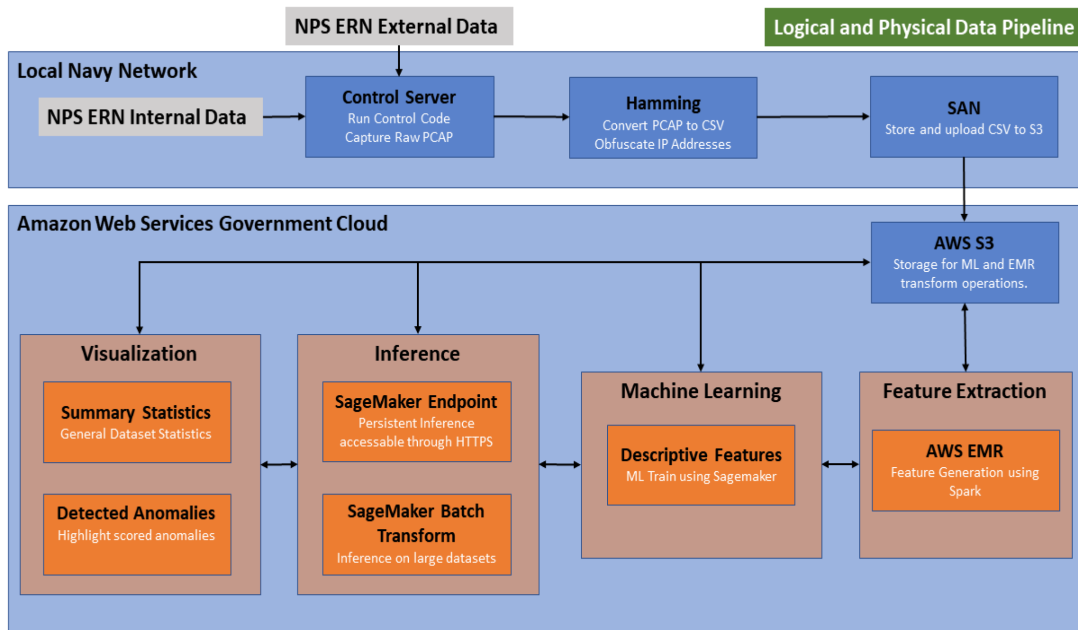

Figure 4.1. Data Flow Topology

4.2 Data Capture

The capture and temporary storage of the dataset as it traversed from the Control Server to S3 was carefully considered and coded for efficiency due to the terabytes of data collected and the computationally heavy operations that were required to take place at each step. Python was chosen as the language to control this process as it has a simple to maintain syntax and can easily access operating system commands while still being relatively portable. Due to the nature of the captured dataset and the network it was recorded from, data-scrubbing requirements from NPS ITACS were complied with, including the obfuscation of IP address network bits. Some basic filtering was conducted in this step to include the removal of all traffic that was not IPv4. The scripts were run from the Control Server and included functionality to transfer the captured data to Hamming for obfuscation of the network bits, conversion to CSV files, and transfer to the Storage Area Network (SAN) for temporary storage before uploading to S3. The code used on Hamming to convert to CSV is proprietary to the implementation of Hamming and not included in an Appendix, however, for reproducibility, it was simply a wrapper for the Wireshark command line utility, TShark, which natively supports conversion to CSV [25].

4.3 Feature Generation

The features that were generated for this pipeline were fundamentally chosen as a proof of concept and were not intended to demonstrate anything beyond the ability of the software to process large volumes of data. Feature generation started in the ‘Hamming’ block, seen in Figure 4.1, where the dataset was reduced from all network traffic, terabytes of data, to just DNS traffic, gigabytes of data. Reduction continued in AWS with application of the Scala programming language in the SparkFeaturePipeline file, found in Appendix E. The dataset size required substantial processing power, leveraged in the form of Apache Spark with a Hadoop powered Elastic MapReduce (EMR) Cluster, to distribute and accelerate the computational load. This processing power was leveraged to remove all rows of the dataset that did not consist of a source IP address from within NPS private IP space with a destination port of 53. AWS EMR Clusters are well suited for this task because they interface well with AWS S3 and SageMaker and, most importantly, can be configured with rules that automatically increase and decrease computational capacity as the load fluctuates, keeping computational costs to a minimum.

The ML model was trained on a subset of the features that were generated. Summary statistics were indexed to individual Source IP addresses, and were computed over the whole dataset to include the same Packet Length Sum, Packet Count, and Average Packet Length as the features that were used in training the ML model. This consolidated the data to a point where processing it with ML would not have provided as accurate of results as possible. This is in part due to the trade-off that comes with time window variance. A smaller time window provides finer granularity but features have a higher variance and vice versa for a larger time window. To prevent this over-consolidation of information, the data was split and processed concurrently in different code branches. A time window based from the recorded Epoch Time feature in the dataset was added to one code branch as shown in Figure 4.2, which had the effect of breaking summary statistics into sections that equal the time window's length. The length of the time window is an easily configurable variable in the source code found in Appendix E and was set to ten minutes for this research. After feature generation, the dataset saw a reduction in size to less than 100 megabytes.

```
// Aggregate sums of packet lengths and counts of packets per time windows.
val intermediateDF = dfWithWindows.groupBy(ip_src.name, time_window).agg(
    sum(col(udp_len.name)).as("Sum_Packet_Length"),
    count(col(udp_len.name)).as("Packet_Count")
)

// Compute and add the average packet lenght per unique source ip per time window to the dataset.
val allFeaturesDF = intermediateDF.withColumn("average_packet_len", $"Sum_Packet_Length" / $"Packet_Count")
```

Figure 4.2. Machine Learning Feature Generation

4.4 Machine Learning

The implementation of RRCF on SageMaker is relatively concise as it depends on objects and API calls as demonstrated in the model creation and training code in Figure 4.3.

```
# specify general training job information and create the rcf object
rcf = RandomCutForest(role=execution_role,
                      train_instance_count=1,
                      train_instance_type='ml.m4.xlarge',
                      data_location='s3://{}/{}/train'.format(bucket, batch_prefix),
                      output_path='s3://{}/{}/output'.format(bucket, ML_prefix),
                      num_samples_per_tree=512,
                      input_mode=input_mode,
                      num_trees=50)

# get the data into the record_set format for rcf
train_data = rcf.record_set(df_train.values, labels=None, channel='train', encrypt=False)

# train on the dataset
rcf.fit(train_data)
```

Figure 4.3. Machine Learning Code

4.5 Inference

After the model is trained, a subset of the data needs to be given anomaly scores, as explained in Section 2.3.3. This process involves taking the validation data, about twenty percent of the total data, and feeding it into the ML model. Twenty percent was chosen because of the quality of visualizations it produces, which would prove the most beneficial to a watchstander. While different ML algorithms will return different results, RRCF will produce and return a simple numerical score per row of the dataset. In this case, each IP Source address received a score for every Time Window it existed in.

When choosing Inference methodologies, two options are readily available within Amazon SageMaker: deploying an endpoint or utilizing Batch Transform jobs. Batch Transform jobs were chosen for their nonpersistent nature and ease of use over more static datasets. Batch Transform also has the built in ability to automatically add the results to the dataset and store it in an S3 bucket. This is seen in Figure 4.4. SageMaker Endpoints may be a better option to explore in future work, which is covered more in depth in Chapter 7.

```
# set up the rcf_transformer as a callable object
rcf_transformer = rcf.transformer(instance_count=1, instance_type='ml.m4.xlarge', output_path=batch_output)

# content_type / accept and split_type / assemble_with are required to use IO joining feature
rcf_transformer.assemble_with = 'Line'
rcf_transformer.accept = 'text/csv'

# Execute the transform job on the batch dataset in the S3 bucket
rcf_transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv',
                          split_type='Line', join_source='Input')
```

Figure 4.4. Batch Transform Code

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results and Discussion

5.1 Data Composition

As described in Section 3.1, the data used for this thesis was not generated but rather collected from a live network on which there was no control over the transmitted content. There was no assumption of preexisting anomalies and no manufactured data that would intentionally stand out as anomalous was injected. As seen in Figure 5.1, the packet sizes show a left distribution normally seen in DNS traffic, which is representative of a standard Navy network.

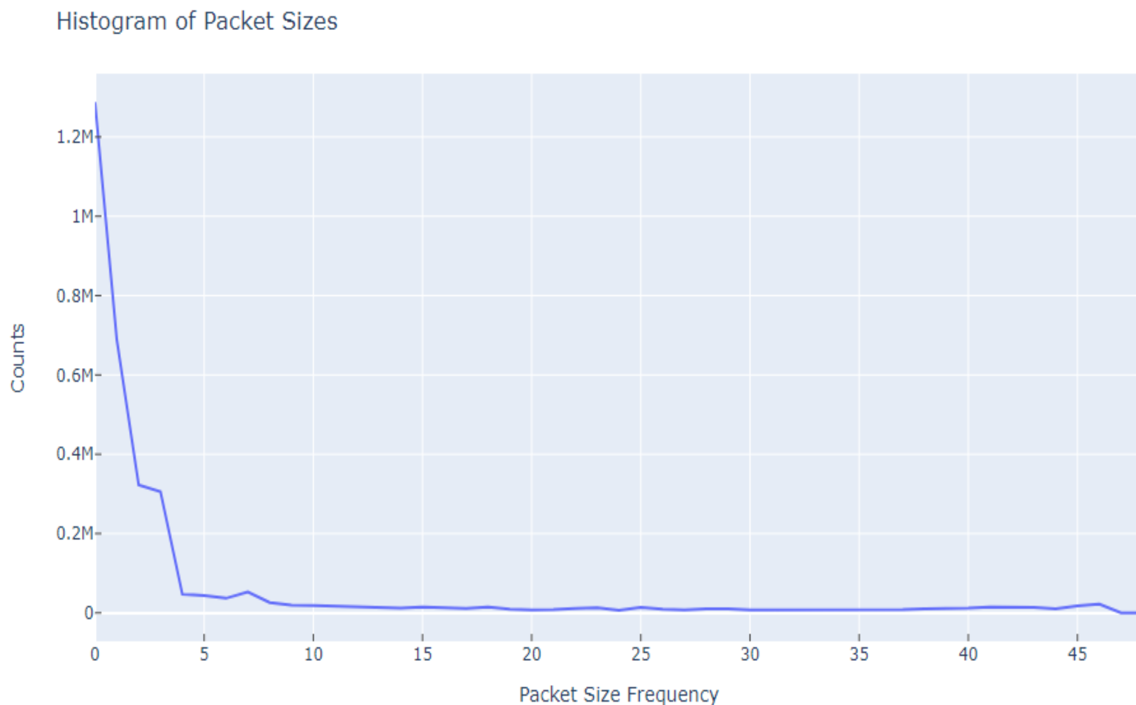

Figure 5.1. Normal Network Traffic

The dataset, for being only about three calendar days in length, was large and diverse. It originally consisted of 1.8TB of PCAP header data, or about 621GB per day. After the

initial conversions and dataset reductions, there was about 6GB per day of DNS packet data in CSV form. Through feature generation, cleaning, and processing discussed in Section 4.3, the remainder was about 9000 unique IP source addresses and the remaining valuable information for about 85 million packets. This information was stored in a variety of different CSV files that were less than 100MB in combined size. The data did not adhere well to the expectation that DNS traffic varies with the normal circadian rhythm of humanity, but had a somewhat consistent average packet size and volume, as shown in Figure 5.2. This might be a factor of dataset length and given enough time to investigate and reduce the outliers through reduction of found network anomalies over time, a circadian rhythm might become more apparent.

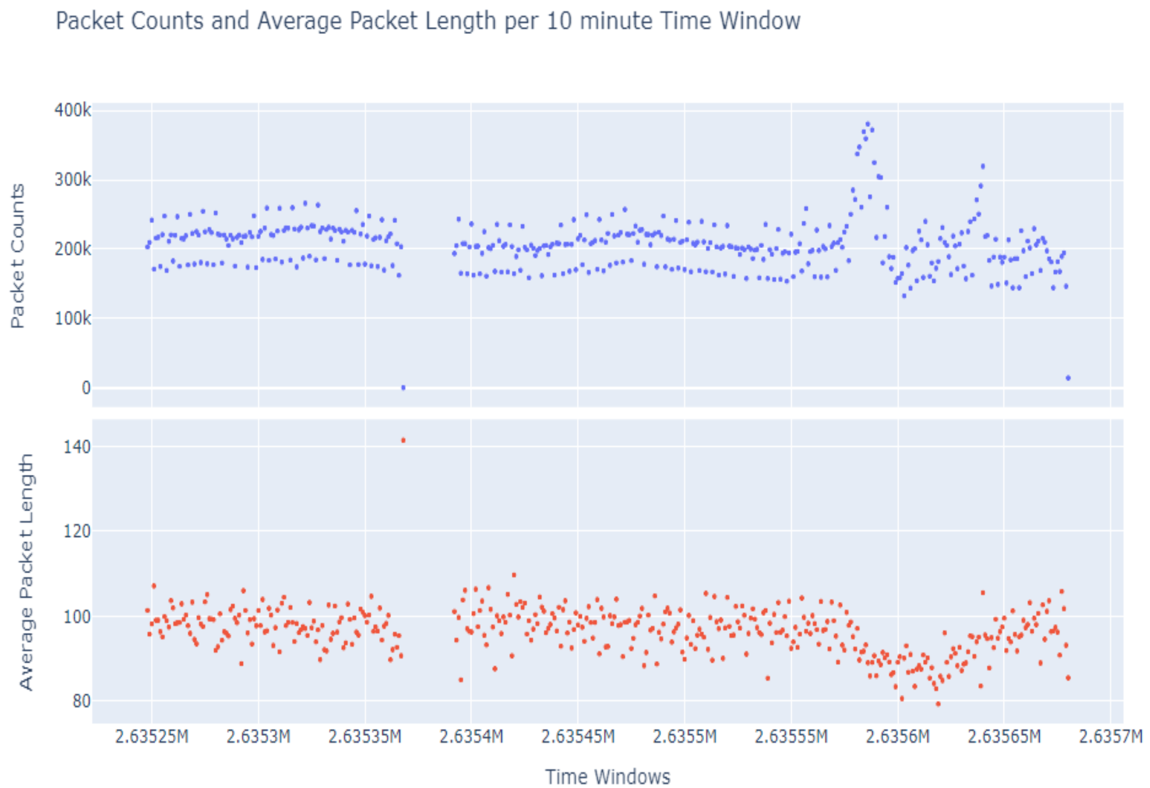

Figure 5.2. Graph of Network Activity over Time

5.2 Data Analysis and Visualization

The final dataset for statistics, visualization, and machine learning consisted entirely of data representing hosts internal to the NPS ERN. As described in Section 4.3, all extraneous information was removed to enable the best employment of the designed features for anomaly detection on this subset of network devices.

Of the three computed features that were presented to RRCF, only the Packet Length Sum feature produced usable results. The Packet Count and Average Packet Length features did not provide a clear separation between anomalous and benign traffic and were discarded. These unused features did not appear to affect the results since RRCF is able to filter extraneous data, as previously discussed.

Figure 5.3, demonstrates the linear nature of how the anomaly scoring occurs over the Packet Length Sum feature. It shows how as the feature increases in value so does the anomaly score, as each point on the graph is indexed by both IP Address and Time Window. From this graph one can see that the anomaly score increasing with packet length sum is consistent with the intuition that many large DNS queries in a short time span should be flagged as anomalous.

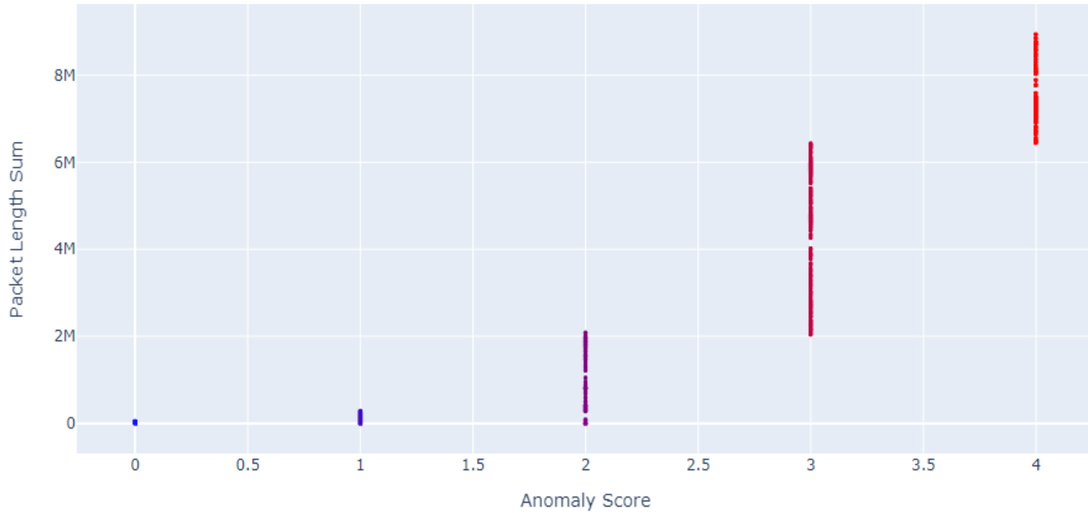

Figure 5.3. Graphical Representation of the Linear Relationship between the Score and Feature

When the Packet Length Sum feature is graphed over the Time Window, and each point is colored by the anomaly score, the linear results remain as the highest scoring anomalies are clustered at the top of the graph, as seen in Figure 5.4. This begins to break the clumps of anomalous data out into individual data points. This is useful but still too dense for a human watchstander to make use of as it is difficult to determine which IP Source addresses are producing the anomalous results.

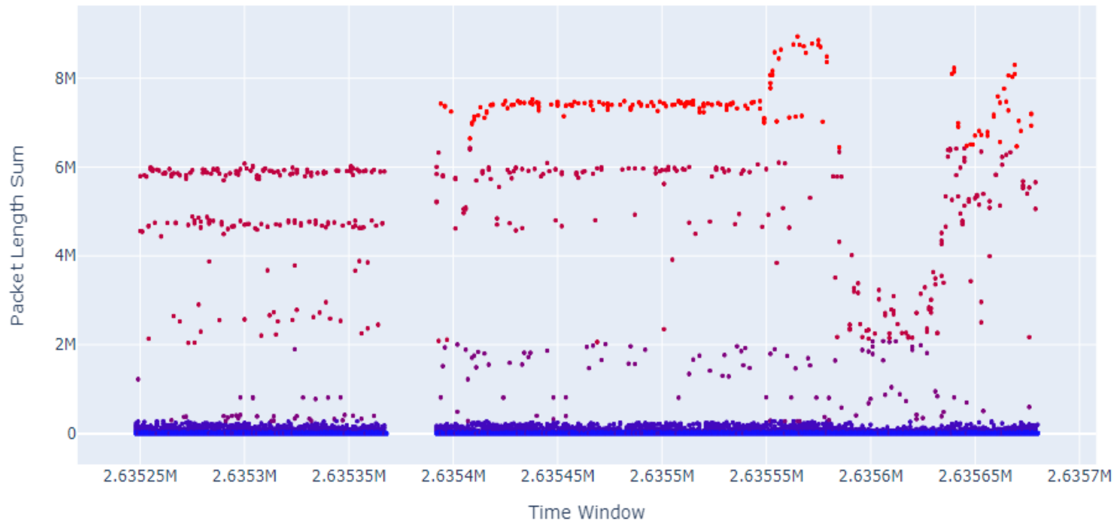

Figure 5.4. Graphical Representation of Feature over Time

Figure 5.5 breaks out IP source address onto its own axis to graphically represent the same data in a way that provides the watchstander a clearer idea of which IP addresses to focus on due to anomalous data originating from those IP sources. The rows seen are anomalous scores showing that the origination point is continuing to produce anomalous activity in each Time Window. As further explained in Section 7.2.6, how to implement this visualisation into a near-real time data feed could be a useful area for future research.

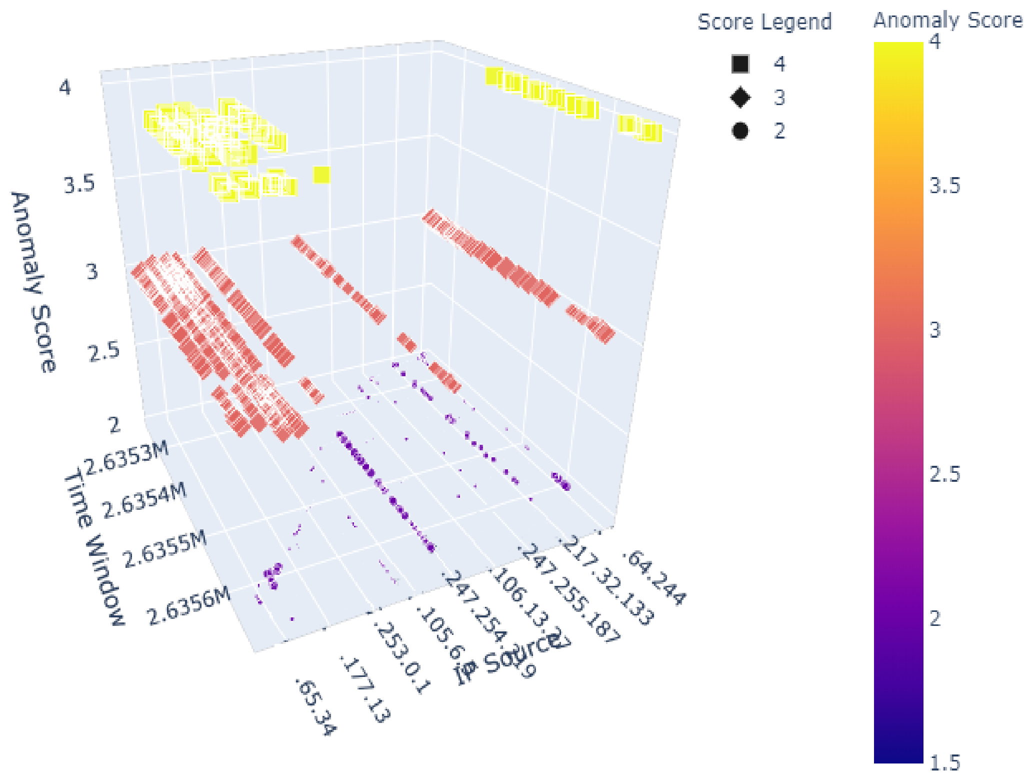

Figure 5.5. Graphical Representation of Time and IP Source by Anomaly

An efficient way to summarize the data for a watchstander or network operator is to give them an easily readable list of IP addresses that have been flagged by the system as anomalous. Figure 5.6 offers this option more clearly than the above image by taking the IP source addresses per time window, aggregating, then normalizing the anomaly scores, and finally displaying the results by IP source address. This graph would be exceptionally useful if updated based on near-real time input as it would show which hosts trend above average.

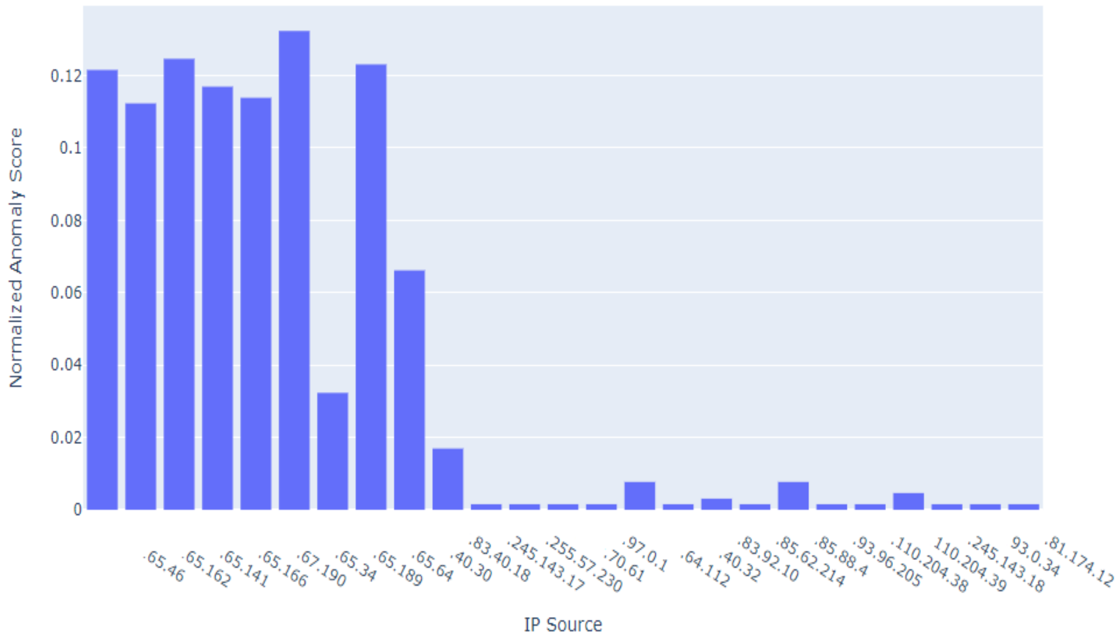

Figure 5.6. Histogram of Anomalous IP Addresses

5.3 Results Discussion

The top two IP addresses were isolated and found to have sent an excessive number of multicast DNS packets over the dataset. NPS ITACS considered these findings to be significant enough that an incident ticket was created to investigate the addresses. It is important to note that because of where the packets were captured on the network, the existing network security tools did not detect the same metrics, but were able to prove that six of the top IP addresses were part of the same functional group of computers. The dataset as recorded and analyzed does not provide the level of granularity to directly allow additional insight into the specific nature of the anomaly, but these results provide a starting point for deeper investigation into anomalous hosts.

This concentration of results shows that as opposed to many anomalous windows across a variety of unrelated IPs, the fact that they are concentrated to only a few related hosts suggests confidence in those hosts actually being anomalous. In other words, RRCF only

trained on individual time winds windows and as such has no sense of how one time window relates to another, but the fact that it consistently picks out the same hosts as anomalous is a strong indication that those hosts are actually behaving out of the norm.

To narrow down the results, the previous graphs in this chapter were created using an anomaly score cutoff of five standard deviations produced by the ML model on the validation dataset which supports an assumption of a normal data distribution. The anomaly score cutoff was determined in this thesis on the basis of data distribution, presentation, and visualization. Testing was conducted for three through ten standard deviations and the results are shown in the following Figure 5.7.

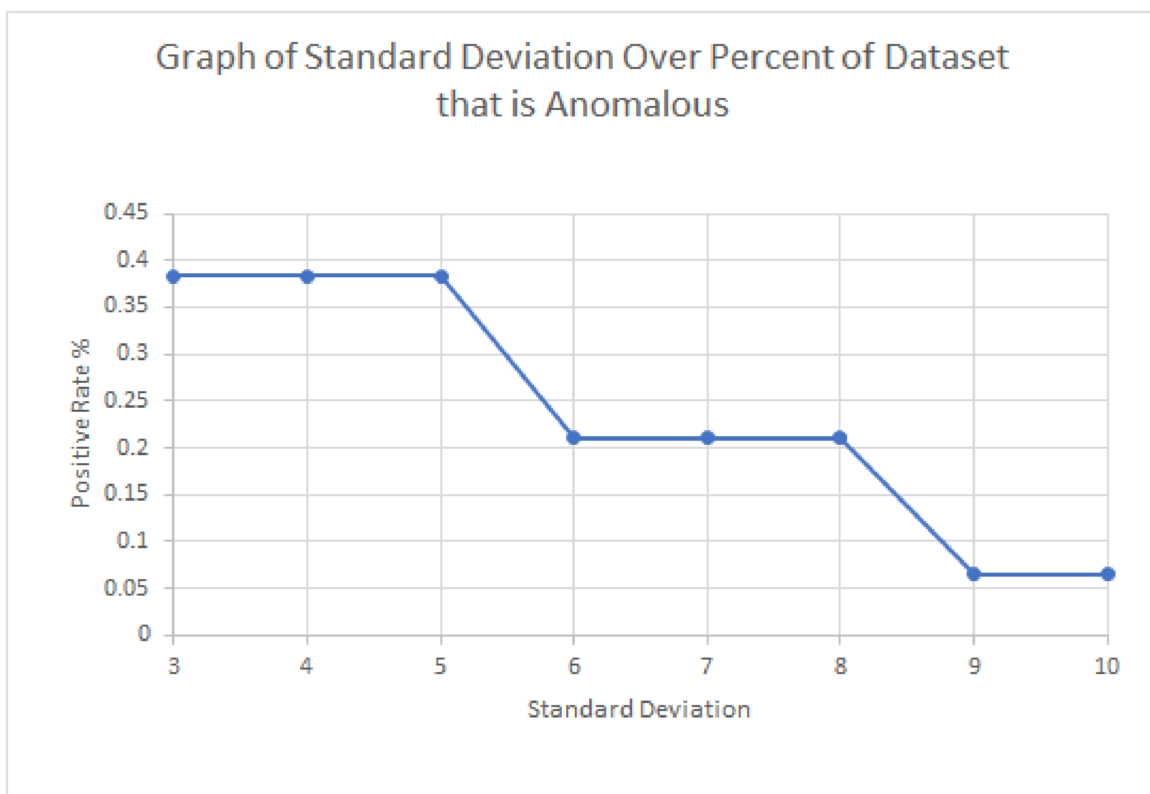

Figure 5.7. Standard Deviation Testing Results

As outlined in the confusion matrix in Section 2.3, if the threshold is set too high the incidence of false negatives would increase and false positives would decrease. This may lead to true anomalous activity being missed by the watchstander. If the threshold is set too

low, the watchstander might become overwhelmed with false positives, leading to normal activity being investigated unnecessarily. If implemented in an architecture like INOSS, the implementation would have to take into account the cost of potentially setting the anomaly score threshold too high or too low. Of note, for some visualizations, setting a higher standard deviation might make a visualization more readable, and for other visualizations, a lower standard deviation will display the information with greater clarity.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6: Implementation within INOSS Framework

6.1 Overview

As part of the initial requirements levied by FCC/C10F, the scope of this thesis is not only to show if ML is effective at detecting anomalies but also to provide a possible method of implementation of the research within existing Navy architecture. The rest of this chapter highlights existing Navy architecture, the architecture used for this research, and how the two could be merged to fit the Navy's needs.

6.2 INOSS

In 2019, a Concept of Operations (CONOPS) was developed by personnel at FCC/C10F that addressed the disjointed and inconsistent nature of how the Navy manages, uses, and protects the DODIN-N, the Navy's portion of the Department of Defense Information Network (DODIN), for Command & Control (C2) and Situational Awareness (SA). The CONOPS sought to mitigate the often slow and generally dysfunctional flow of information from the edge of the networks to the watchstander on a watchfloor. The operating model that came out of this broad discussion about how to conduct business more efficiently is the INOSS Framework shown in Figure 6.1. This is the method by which FCC/C10F will "operate, maintain, secure, protect, defend, and maneuver the DODIN-N" [26]. The INOSS Framework provides a standardized structure for all of the Navy's existing cyber tools and capabilities to fit within. Once under the umbrella of INOSS, any overlap of similar tools or capabilities can be merged and any gaps can be quickly identified and addressed.

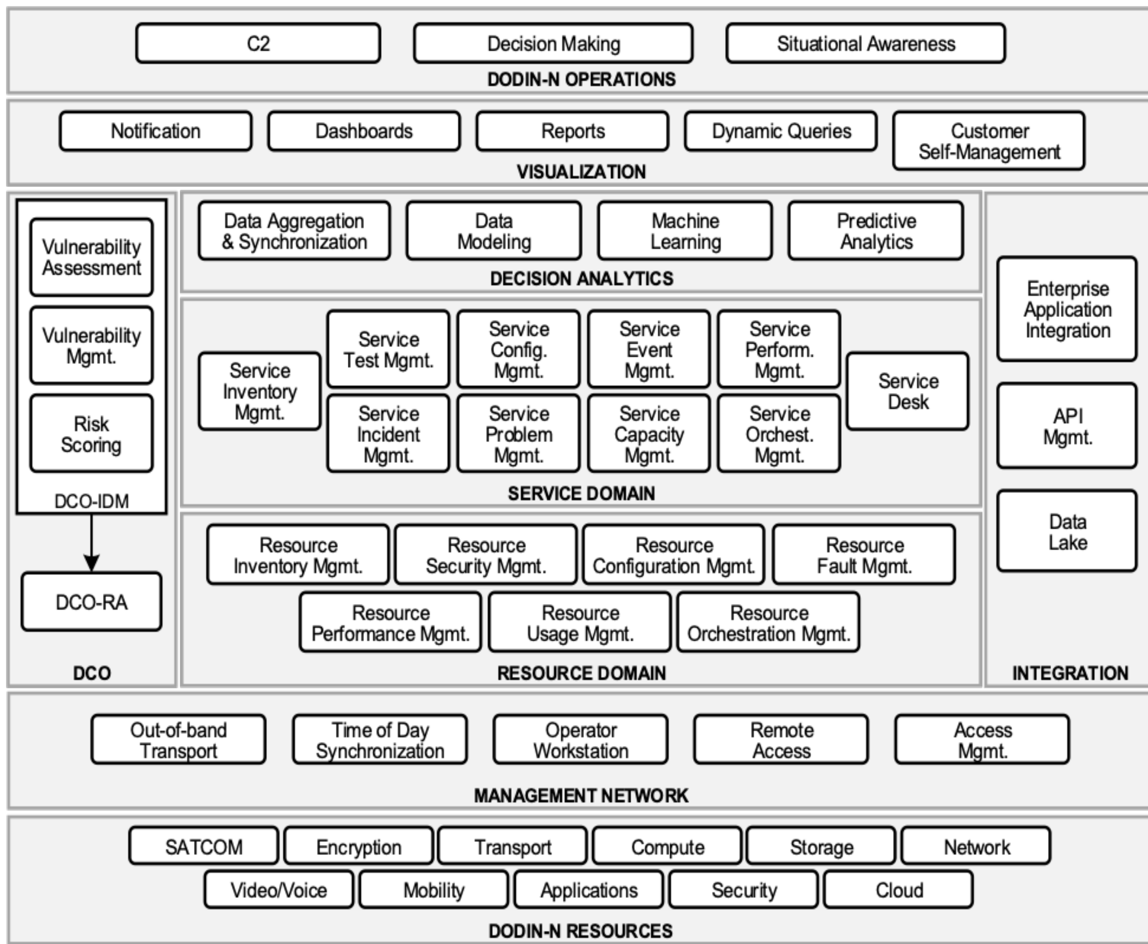

Figure 6.1. INOSS Framework. Source: [26].

Using ML to detect network anomalies fits within the framework under Decision Analytics. Decision Analytics is making decisions using data science principles on data that is collected throughout the DODIN-N. The use of data science to analyze and make decisions increases the efficiency and effectiveness with which responses and actions can be taken on the network. Decision Analytics will directly drive visualization and help determine what the watchstanders will see. In the case of anomaly detection, a visual representation of the anomaly detected will be pushed to the watchfloor for SA. As seen in Figure 6.2, ML will be applied in conjunction with data modeling, predictive analytics, and data aggregation and synchronization.

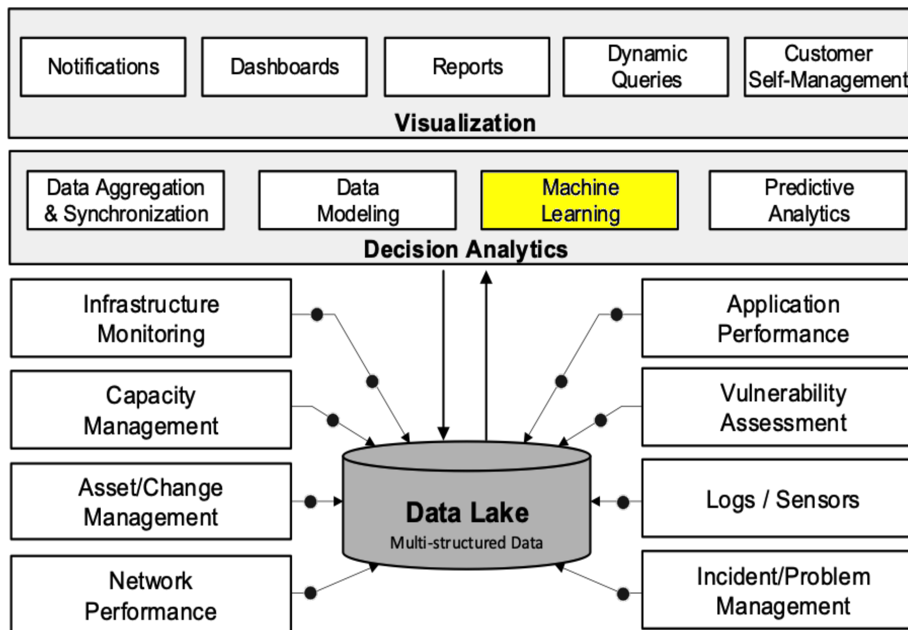

Figure 6.2. Decision Analytics. Source: [27].

6.3 AWS

AWS is a comprehensive suite of tools hosted on servers owned by Amazon, that are accessible globally. A variety of different services are available but the one used to build and train the ML model was Amazon SageMaker. In order for Amazon SageMaker to be accessed, a connection with AWS must first be established. The user is then free to access Amazon SageMaker, as well as any of the other services currently offered by Amazon. What this means for the end user is that they do not have to spend the money on infrastructure, maintenance, security, upkeep, and upgrades (e.g., memory, storage, etc.) that would typically be cost prohibitive for a large company, let alone a single end user.

6.3.1 Amazon SageMaker

Amazon SageMaker is a ML service and API offered by Amazon that enables a user to build, train, test, deploy, and evaluate a ML model. Another benefit is the ability for multiple people to access the notebooks with code, allowing for collaborative efforts. Additionally, the code from Amazon SageMaker notebooks can be transferred for local use, allowing a

company to make use of their own infrastructure to train, test, and deploy a ML model [28]. SageMaker was the primary choice to conduct ML for this research due to the fact that it is already integrated into an infrastructure that can handle large amounts of data, is automatically scalable, and has built in ML algorithms, omitting the need to create custom algorithms. The AWS costs for the research on a static dataset were negligible and a realistic cost analysis of large scale operation remains an item for future work.

6.4 Amazon SageMaker within INOSS

Perhaps the most cost effective and quickest option for implementing ML within the INOSS framework is to use AWS. This approach would be more efficient and effective from the perspective of time required to develop and maintain an architecture since AWS is already a robust managed infrastructure that is easily able to automatically adapt to accommodate large volumes of data. AWS maintains and updates their servers regularly which means the Navy will always have access to the latest hardware and software versions, to include all security patches. The Navy is organized such that the vast majority of the Navy's fleet data flows through fleet NOCs and therefore these are the ideal locations to deploy the ML models. The downside, as aforementioned, is that AWS requires connectivity to use. The fleet NOCs would have to be able to sustain a connection to the AWS servers in order to train new models. This could prove challenging, especially in a denied environment that could arise during a conflict with an adversarial nation.

The second option for adopting ML within the Navy's existing INOSS framework would be to train and deploy models using Amazon SageMaker, or a similar ML API, and then run those models locally. The primary hurdle to overcome with a custom built infrastructure would be ensuring that the storage and processing capacity was high enough to account for the sheer volume of data that would flow through the servers. Not only would there be a high volume of data, the processing power would need to account for deploying the model over streaming data in a real-time environment. In addition, if the Navy were to contract out the development, installation, maintenance, and upgrade cycle of the infrastructure, it would potentially be outdated by the time it was installed.

The final option would be to build and produce everything local from beginning to end. This would include acquiring a Navy owned API, creating models with this API locally,

and then deploying locally as well. The same obstacles as mentioned in option two would apply to this, with the additional difficulty of contracting and obtaining a Navy specific API. Figure 6.3 shows a comparison of the first and last options.

Infrastructure Comparison		
	AWS	**Local**
Cost	No upfront cost for development but would pay a service fee.	Overall costs would likely exceed that of AWS due to need for contract to build out infrastructure and cost of hardware and software.
Ease of Implementation	Easier to implement than local infrastructure because the hardware and software are already built out and ready for use.	A local infrastructure would have to go through the Navy contracting cycle which could take a year or more.
Maintenance and Upkeep	Maintenance and upkeep are conducted by AWS and this would be included in the service fees.	Maintenance and upkeep could be conducted by military personnel stationed at the NOC. There would be no extra cost associated with it.
Technological Upgrades	AWS conducts upgrades as more compute power comes available. The cost for using the larger and more powerful servers is higher, however the older, less powerful servers remain available for less cost.	The Navy would need to either include technological upgrades in an ongoing contract or renegotiate new contracts when more powerful technology comes on the market. Using COTS would be the ideal solution for a local infrastructure.
Connectivity Requirements	Because AWS is a cloud based service, internet connectivity would be required to build new models based on streaming real-time data.	Because the infrastructure would be local to each NOC, there would be no connectivity requirement to use the servers for ML.

Figure 6.3. Infrastructure Comparison

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7: Conclusion and Future Research

7.1 Overview

The results from Chapter 5 ultimately showed that ML can in fact be used to detect anomalies within a Navy dataset. The caveat to this is that the proof-of-concept is only valid at this point as it relates to DNS traffic. The rest of this chapter outlines some of the issues faced as we collected data, created a pipeline, and ran ML on the dataset. It also provides areas for future work, key takeaways from the research, and ends with a conclusion.

7.2 Thesis Limitations and Future Research

The course of this thesis changed dramatically over its existence. The data source, data collection, coding, dataset, and data processing platforms all underwent various changes throughout the development process. Throughout this process much was learned, especially when it comes to identifying areas that still need to be explored.

7.2.1 Data Source

Originally we sought to collect data from an afloat Navy platform. This would have been the best application of this applied research proof-of-concept because the shipboard networks are highly controlled. Unlike the NPS network, they do not allow the users to bring their own devices and as such would have allowed for easier data analysis. The shipboard dataset would have been significantly smaller, for even though NPS utilizes only ten percent of its available bandwidth, that still is greater than the maximum capacity of the average shipboard connection by a factor of a thousand. The thesis team was unable to gain authorized access to afloat data in the time frame required, leaving this bureaucratic task to be solved by future researchers.

7.2.2 Data Collection

Because we collected on a larger network, the need to capture all the data, as opposed to just a sample, for this applied research project pushed existing hardware beyond its normal limits, which caused significant problems. For example, the data collection hardware was initially insufficient, making data loss a recurring issue and infrastructure had to be modified or completely reworked on both the hardware and software levels several times to support the volume of data. Different functions of the logical and physical data pipeline, referenced in Figure 4.1, shifted between devices and locations as necessary to reduce this data loss and increase processing speed as much as possible. These efforts significantly delayed any substantial data capture efforts.

Future research in anything regarding the use of IP data can leverage the data collection architecture that we developed in conjunction with ITACS on the NPS campus. The data pipeline that was developed is robust, capable of scaling up to support significant volumes of sustained network traffic, and has potential to serve a plethora of research projects.

7.2.3 Code

The selection of programming languages was a process of trial and error. The final selections of Scala for feature generation and Python for ML and visualization as described in Section 3.4 were chosen in response to difficulties working with other languages. For feature generation, the difficulties were primarily based on the size of the data and the speed at which processing needed to occur. We did not find a language that had the same feature set as Scala which also incorporated a Spark interface to the compute clusters. The native support of Scala on AWS and its connection to the HDFS that runs on EMR clusters, was key in our goal of creating a scalable solution. Python was the language of choice for ML and visualization because of the API availability and support. The code in the appendices may serve future researchers as a foundation for data collection, transformation, feature generation, machine learning, inference, and visualization.

7.2.4 Dataset

Only a small fraction of the collected data was used through the whole data pipeline. Most was removed early in the process, but could easily be retained for use in later steps. In this thesis the research was conducted only on DNS traffic, but adding other protocols could

show that ML can find anomalies other than ones that are DNS related. This would be especially important if using shipboard data, as the afloat network would produce different results than a shore network based on the differences in the shape of the data, which might include variances in protocol distribution or different protocols entirely. When adding a protocol, the research would then need to expand the feature generation to include this new information as the feature pipeline that the thesis team created is only functional for DNS traffic.

Another area for future research with regard to the dataset is using the injection of known anomalies in order to better quantify the effectiveness of the ML model. If this were done building on the use of just DNS traffic then the model could be trained on normal data and then tested with known anomalies. To inject known anomalies, a tool such as Iodine could be used to create a covert channel within the network and the resulting anomalies should be detected when testing the ML model. This, if conducted under controlled conditions, would create a dataset that could result in semi-supervised model evaluation.

7.2.5 Data Processing Platform

The AWS platform is powerful, flexible, and has a somewhat user friendly interface allowing the use of significant, if potentially costly, computing resources. A potential downside to utilizing AWS is the rapid rate of change on the platform and as a result, documentation is sometimes out of date. This was encountered by the thesis team somewhat often on AWS's relatively new SageMaker ML platform. It is important to note this research was conducted on the AWS GovCloud, which is a older version of the publicly available AWS with a different software package update schedule. The platform is flexible, powerful, and is able to handle any size data set at scale, but with those benefits, it comes with a steep learning curve that does not always have documentation to help the new user, thus configuration of the connections between all the data pipeline segments took significant time and research.

The flexibility and available options of the AWS SageMaker ML platform cannot be understated. Due to constraints, this thesis utilized the AWS SageMaker Batch Transform option to infer on the dataset, but a better solution for a watchfloor environment is the implementation of a persistent endpoint that can automatically produce interference results on a near real-time basis. This would allow for the automation of a major part of the data pipeline

and create the ability to run the ML on streaming data which create many opportunities for future research.

7.2.6 Visualization for the Watchstander

The other area that this research project begins to explore, but that future work is needed in, is the visualization of how the watchstander is able to not only view the data, but also interact with this visual representation. Allowing the watchstander to interact with the visualization could provide them much needed meaning and value on different aspects of the results. For example, there is currently no way to manipulate the time window within the visualization when looking at anomalies by unique IP address. Creating a mechanism to manipulate this could allow a watchstander to scroll back in time to quickly determine if a unique IP address has multiple types of anomalies. To give the watchstander the easiest and most functional interactive experience, future work could implement the above in a near-real time data feed to increase responsiveness.

7.3 Key Takeaways

"Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it..." [29].

This quote illuminates the lack of resources available to accomplish research at the scale attempted by this applied research project. While big data is often discussed, few do more than talk. From the data collection pipeline, to data wrangling, and to running ML on the resulting data, there were few references or resources on how to conduct this project on the terabytes of data that are flowing through the NPS ERN on a daily basis. Solutions to big data problems tend to be very problem specific however, so future research will likely have its own hurdles to overcome.

Problem solving occurred at every level: creating infrastructure, enabling accurate packet capture, ensuring acceptable compute times for every step of the data pathway. Despite the obstacles faced, one potential takeaway from this thesis could be that ML is a viable option for the Navy to support existing IDSs and IPSs. This research serves primarily as a proof-of-concept, showing that a data pipeline can be built and that ML can be used to detect network traffic anomalies.

APPENDIX A: Data Control Code

```
#/*************************************************************
#*File Name: control.py
#*Description: Used to control and schedule the scripts for
#capture, obfuscation, and transfer.
#* Language: Python
#*Authors: CTNC Jackie Turner (jackie.turner@nps.edu),
#Dr. Vinnie Monaco (vinnie.monaco@nps.edu), LT Michael Laws
#(michael.laws@nps.edu), LT Greg Bunder (greg.bunder@nps.edu)
#*************************************************************/

#!/usr/bin/python3
import os
from subprocess import Popen, PIPE

# Note: modify only these to control interfaces, pcap location, destination
# directory, and length of capture
INSIDE_INTERFACE = 'eth0' # NPS specific values scrubbed
OUTSIDE_INTERFACE = 'eth0' # NPS specific values scrubbed
PCAP_DIR = '/tmp/pcap/'
LOG_FILE = '/tmp/pcap/pcap.log'
TRANSFER_TO = '/folder/' # NPS specific values scrubbed
CAPTURE_START = 'midnight'

# all units are seconds
CAPTURE_DURATION = 60*60*24*7 # 7 days
BUFFER_DURATION = 60*60 # 1 hour
TRANSFER_INTERVAL = 60*60*8 # 8 hours
OBFUSCATE_INTERVAL = 60*60*24 # 24 hours

if __name__ == '__main__':
    os.makedirs(PCAP_DIR, exist_ok=True)
    f = open(LOG_FILE, 'a')

    # capture at midnight
    Popen([
```

```
36            'at',
              CAPTURE_START,
38        ], stdout=f, stderr=f, stdin=PIPE).communicate(input='python3 capture.py'.encode())

40    # transfer every t//60 minutes. at doesn't support seconds
      # and allow a grace period of 2*t//60 after capture ends
42     transfer_interval_min = TRANSFER_INTERVAL//60
      capture_duration_min = CAPTURE_DURATION//60
44    for i in range( transfer_interval_min ,
                      capture_duration_min + 2* transfer_interval_min ,
46                     transfer_interval_min ):
          Popen([
48            'at',
              f'{CAPTURE_START} + {i} minutes',
50        ], stdout=f, stderr=f, stdin=PIPE).communicate(input='python3 transfer.py'.encode())

52    # regularly submit jobs to obfuscate
      obfuscate_interval_min = OBFUSCATE_INTERVAL//60
54    for i in range( obfuscate_interval_min ,
                      capture_duration_min + 2*obfuscate_interval_min ,
56                     obfuscate_interval_min ):
          Popen([
58            'at',
              f'{CAPTURE_START} + {i} minutes',
60        ], stdout=f, stderr=f, stdin=PIPE).communicate(
          input='ssh user@server "srun -n1 -t600 -lobfuscate --mem=64gb /folder/obfuscate.py"'.
      encode())   # NPS specific values scrubbed
```

APPENDIX B:
Data Capture Code

```
#/*************************************************************
#* File Name: capture.py
#* Description: This script is used to run an at job every 24
#hours that will start and end a capture on interfaces em1 and
#em2.
#* Language: Python
#* Authors: CTNC Jackie Turner (jackie.turner@nps.edu),
#Dr. Vinnie Monaco (vinnie.monaco@nps.edu), LT Michael Laws
#(michael.laws@nps.edu), LT Greg Bunder (greg.bunder@nps.edu)
#*************************************************************/

#!/usr/bin/python3
import os
import time
from subprocess import Popen
from control import *

NOW = time.strftime("%Y.%m.%d")

# capture settings
SNAPLEN = '70'
MEM_BUFFER = '4096'

# locations of things
TSHARK = 'tshark'
INSIDE_DIR = f'{PCAP_DIR}/inside/{NOW}'
OUTSIDE_DIR = f'{PCAP_DIR}/outside/{NOW}'
INSIDE_PREFIX = f'{INSIDE_DIR}/inside'
OUTSIDE_PREFIX = f'{OUTSIDE_DIR}/outside'

if __name__ == '__main__':
    os.makedirs(INSIDE_DIR, exist_ok=True)
    os.makedirs(OUTSIDE_DIR, exist_ok=True)

    f = open(LOG_FILE, 'a')
```

```
36
    Popen([
38      TSHARK,
        '-i', INSIDE_INTERFACE,
40      '-f', 'ip', # filter ipv4
        '-s', SNAPLEN,
42      '-B', MEM_BUFFER,
        '-b', f'duration:{BUFFER_DURATION}',
44      '-a', f'duration:{CAPTURE_DURATION}',
        '-F', 'pcap',
46      '-w', INSIDE_PREFIX,
    ], stdout=f, stderr=f)
48
    Popen([
50      TSHARK,
        '-i', OUTSIDE_INTERFACE,
52      '-f', 'ip', # filter ipv4
        '-s', SNAPLEN,
54      '-B', MEM_BUFFER,
        '-b', f'duration:{BUFFER_DURATION}',
56      '-a', f'duration:{CAPTURE_DURATION}',
        '-F', 'pcap',
58      '-w', OUTSIDE_PREFIX,
    ], stdout=f, stderr=f)
```

APPENDIX C: Data Transfer Code

```
#/************************************************************
#* File Name: transfer.py
#* Description: This script is used to move files to Hamming.
#* Language: Python
#* Authors: CTNC Jackie Turner (jackie.turner@nps.edu),
#Dr. Vinnie Monaco (vinnie.monaco@nps.edu), LT Michael Laws
#(michael.laws@nps.edu), LT Greg Bunder (greg.bunder@nps.edu)
#************************************************************/

#!/usr/bin/python3
from subprocess import call
from control import *

if __name__ == '__main__':
    f = open(LOG_FILE, 'a')

    # first do an rsync
    call([
        'rsync',
        '-h', # human readable numbers
        '-v', # verbose
        '-r', # recurse into directories
        '-t', # preserve modification times
        '-P', # --partial (keep partially transferred files) + --progress (show progress during transfer)
        '--remove-source-files', # remove source files after successful transfer
        '--exclude="*.log"',
        PCAP_DIR,
        TRANSFER_TO,
    ], stdout=f, stderr=f)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: Data Obfuscation Code

```
#/*************************************************************
#* File Name: obfuscate.py
#* Description: This script is used to obfuscate the internal
#IP addresses.
#* Language: Python
#* Authors: CTNC Jackie Turner (jackie.turner@nps.edu),
#Dr. Vinnie Monaco (vinnie.monaco@nps.edu), LT Michael Laws
#(michael.laws@nps.edu), LT Greg Bunder (greg.bunder@nps.edu)
#*************************************************************/

#!/usr/bin/python3
import glob
from subprocess import call

TCPREWRITE      = '/folder/' # NPS specific values scrubbed
LOG_FILE        = '/folder/' # NPS specific values scrubbed
SOURCE          = '/folder/' # NPS specific values scrubbed
DESTINATION     = '/folder/' # NPS specific values scrubbed

if __name__ == '__main__':
    f = open(LOG_FILE, 'a')

    source_outside = os.path.join(SOURCE, 'outside')
    source_inside = os.path.join(SOURCE, 'inside')
    destination_outside = os.path.join(DESTINATION, 'outside')
    destination_inside = os.path.join(DESTINATION, 'inside')
    os.makedirs(destination_outside, exist_ok=True)
    os.makedirs(destination_inside, exist_ok=True)

    # outside files: move to destination
    call([
        'rsync',
        '-h', # human readable numbers
        '-v', # verbose
        '-r', # recurse into directories
```

```
36          '-t', # preserve modification times
            '-P', # --partial (keep partially transferred files ) + --progress (show progress during
        transfer )
38          '--remove-source-files', # remove source files after successful transfer
            '--exclude="*.log"',
40          source_outside ,
            destination_outside ,
42      ], stdout=f, stderr=f)

44      # inside files : obfuscate and move if success
        for infile in glob.glob(os.path.join(SOURCE, 'inside/*/inside_*')):
46          day, fname = infile . split ('/')[-2:]
            outdir = os.path.join( destination_inside , day)
48          os.makedirs( outdir , exist_ok=True)
            outfile = os.path.join( outdir , fname)
50
            exitval = call ([
52              TCPREWRITE,
                '--pnat=x.x.x.x/x:x.x.x.x/x,x.x.x.x/x:x.x.x.x/x' # NPS specific values scrubbed
54              '--infile=' infile ,
                '--outfile=' outfile ,
56              '--fixcsum',
            ], stdout=f, stderr=f)
58
            if exitval == 0:
60              os.rm( infile )
            else :
62              print ('Error obfuscating :', infile , file =f)
```

APPENDIX E: Spark Feature Pipeline Code

```
/***************************************************************
* File Name: SparkFeaturePipeline
* Description: Ingest, transform, and export CSVs as formatted
by the pipeline.
* Language: Scala
* Author: LT Michael Laws (michael.laws@nps.edu)
* Contributor: LT Greg Bunder (greg.bunder@nps.edu)
* Thesis Advisor: Dr. Vinnie Monaco (vinnie.monaco@nps.edu)
***************************************************************/

// Imports
// Adds necessary Apache Spark and Scala Libraries
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.types._
import org.apache.spark.sql.expressions._
import org.apache.spark.sql.expressions.UserDefinedFunction
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions.udf
import scala.math.floor
import java.net.InetAddress
import org.apache.hadoop.conf._

// Parameter Definitions
// Configurable parameters to allow for ease of changing program functionality

val window_size = 600 // Set the window size, integer in seconds. 600 seconds = 10 minutes
val time_window = "time_window"

// Information to remove DNS Server IP address information from Data
// servers to remove in IP address form, ie, "1.21" would be an acceptable host bit string to
    denote a IP source in the following sequence
// string read from the right of the IP address
```

```
val serverIPSeq= Seq("1.121" ) // NPS specific values scrubbed

// base network value for internal traffic .
// strings contain characters read from the left , ie "192."
val internalTraffic = Seq("192.") // NPS specific values scrubbed

// Set up options to pull files from S3.
// load whole data set
val dataSet = "s3 :// bucket/ prefix /.*/*. csv" // NPS specific values scrubbed

/**********************************
// S3 buckets to write CSVs to
**********************************/

// ML File:
val packetFeaturesPath = "s3 :// bucket/ prefix " // NPS specific values scrubbed

// Visualization files :
val summaryStatsPath = "s3 :// bucket/ prefix " // NPS specific values scrubbed
val packetHistPath = "s3 :// bucket/ prefix " // NPS specific values scrubbed
val TimeWindowSummaryStatsPath = "s3://bucket/prefix" // NPS specific values scrubbed


// Dataset Column Definitions
// Each column is defined as a data type and set as an immutable value
// Set up Structs for each column in the dataset
val frame_time_epoch = StructField ("frame_time_epoch", DoubleType, true )
val ip_src = StructField (" ip_src ", StringType , true )
val ip_dst = StructField (" ip_dst ", StringType , true )
val udp_len = StructField ("udp_len", IntegerType , true )
val udp_srcport = StructField ("udp_srcport", IntegerType , true )
val udp_dstport = StructField ("udp_dstport", IntegerType , true )

/**********************************
// Functions and User Defined Functions
// All required compute functionality to support feature generation
**********************************/

// Declare the schema of the columns in the dataset as a struct type
val SCHEMA = StructType(Seq(frame_time_epoch, ip_src, ip_dst , udp_len, udp_srcport , udp_dstport
```

```
    )
)

// Use the SCHEMA val in a simple function to load the dataset into a dataframe.
def read(path: String): DataFrame = {
    spark.read.format("csv").option("header", "true").option("delimiter", ",").schema(SCHEMA
    ).load(path)
}

// time window compute function
def computeWindow(time: Float, window_size: Integer): Integer = {
    time.toInt / window_size
}

// time window UDF
val computeWindowUDF = udf{(time: Float) => computeWindow(time, window_size)}

// basic conversion functions
def bool2int(b:Boolean) = if (b) 1 else 0
val toInt    = udf[Int, Double]( _.toInt)
val toDouble = udf[Double, Int]( _.toDouble)

// User defined function to compute the size of the buckets based on the Freedman-Diaconis rule
// http://bayes.wustl.edu/Manual/FreedmanDiaconis1_1981.pdf
// https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.650.2473
def computeBucketSize(data: DataFrame, colName: String): Double = {
    val n = data.count
    val quantiles = data.stat.approxQuantile(colName, Array(0.25, 0.75), 0.25)
    2.0 * (quantiles(1) - quantiles(0)) / math.pow(n, 1.0/3)
}

// For Histogram data, compute the min and max buckets
def computeBucketEndPoints(data: DataFrame, colName: String): Seq[Double] = {
    val bucketSize: Double = computeBucketSize(data: DataFrame, colName: String)
    val minMaxVals = data.agg(min(colName), max(colName)).rdd.take(1)(0)
    val minVal = minMaxVals.getDouble(0)
    val maxVal = minMaxVals.getDouble(1)
    val numBuckets: Integer = math.ceil((maxVal - minVal)/ bucketSize).toInt
    val leftBucketEndPoints: Seq[Double] = (0 until numBuckets).map(_ * bucketSize + minVal).
    toSeq
```

```
        leftBucketEndPoints
114 }

116 // For Histogram Data, map the rows to the correct bucket
    def mapValueToBucketIndex(value: Double, leftBucketEndPoints: Seq[Double]): Integer = {
118     val numBuckets = leftBucketEndPoints.length
        def leftInclusiveIndex (i: Integer = 0): Integer = {
120         if (i >= numBuckets - 1) numBuckets - 1 // return last bucket index
            else if (leftBucketEndPoints(i) <= value && value < leftBucketEndPoints(i + 1)) i
122         else leftInclusiveIndex (i + 1)
        }
124     leftInclusiveIndex (0)
    }
126
    // For Histogram Data, udf to map row values to bucket index
128 def mapValueToBucketIndexUDF(leftBucketEndPoints: Seq[Double]): UserDefinedFunction = {
        udf{v: Double => mapValueToBucketIndex(v, leftBucketEndPoints)}
130 }

132 // Next three functions are used to remove DNS servers from the dataset
    // using string matching from the right side of the string for the length of the string to be
        matched.
134
    // return a boolean, true if the strings match, false otherwise
136 def ipFilter (s: String, endPart: String): Boolean = {
        s.takeRight((endPart.length)) == endPart
138 }

140 // returns true if any string matches any part of the sequence
    def ipFilterSeq (s: String, endPartSeq: Seq[String]): Boolean = {
142     val intSeq = endPartSeq.map(ep => bool2int( ipFilter (s, ep)))
        intSeq.sum > 0
144 }

146 // get a boolean to match strings udf section of above
    def ipFilterSeqUDFfactory(endPartSeq: Seq[String]): UserDefinedFunction = {
148     udf{(s: String) => ipFilterSeq (s, endPartSeq)}
    }
150
    // Next three functions are used to remove non-internal traffic from the dataset
```

```
152 // using string matching from the left side of the string

154 // return a boolean, true if the strings match, false otherwise
    def beginingStringFilter (s: String , startString : String ): Boolean = {
156     s. startsWith ( startString )
    }
158
    // returns true if any string matches any part of the sequence
160 def stringFilterSeq (s: String , stringSeq : Seq[ String ]) : Boolean = {
        val intSeq = stringSeq .map(ep => bool2int( beginingStringFilter (s , ep)))
162     intSeq .sum > 0
    }
164
    // get a boolean to match strings udf section of above
166 def beginingStringSeqUDFfactory( stringSeq : Seq[ String ]) : UserDefinedFunction = {
        udf{(s: String ) => stringFilterSeq (s , stringSeq )}
168 }

170 /********************************
    //Load the Dataset into the DataFrame
172 ********************************/

174 // Uses the read function to load the dataset into a dataframe
    // Read in the CSV files
176 val df = read( dataSet )

178 // Remove Null Values
    // Removes all irrelevant null values , outputs a count of the original
180 // and post removal rows, and computes the percentage of rows lost

182 // count all and nulls in dataset
    val originalCount : Float = df .count ()
184 val countNull = df .where(' ip_src . isNull ) . groupBy(' ip_src ) . count ()

186 // remove all rows with null values from the dataset
    val pstDrop = df .na.drop ()
188 val df = pstDrop // must seperate this line with the above ...
    val postNullRemove: Float = df .count ()
190 // Calculate the percent dropped based on null values in rows
    val percentDrop: Float = (( originalCount − postNullRemove) ∗ 100) / originalCount
```

```
192
    // Remove DNS Servers
194 //The shape of the traffic originating from DNS servers is very different
    // from what originates from normal network users. This cell removes all
196 // IP addresses, ie, DNS Servers and their traffic, that are part of the
    // sequence declared in the Parameter cell.
198
    // drop DNS servers from the data
200
    // Create a UDF val from the UDF factory to apply to the dataset
202 val DNSServerFilterUDF: UserDefinedFunction = ipFilterSeqUDFfactory(serverIPSeq)

204 // Remove all true rows of the column "isDNSserver" from the dataset
    val noDNSServers = df.withColumn("isDNSserver",
206                             DNSServerFilterUDF(col(ip_src.name))).filter(!col("isDNSserver"
      ))

208 // Drop the unneeded boolean column from the dataset
    val df = noDNSServers.drop("isDNSserver")
210
    // Remove Non-internal traffic
212 //The focus of the feature generation is detecting anomalies originating
    // from hosts within the internal network. If the source IP address is not
214 // from a internal address, then we are assuming that it is not a host on
    // the network and removing it in this step.
216
    // Create a UDF val from the UDF factory to apply to the dataset
218 val networkFilterUDF: UserDefinedFunction = beginingStringSeqUDFfactory( internalTraffic )

220 // Remove all false rows of the column " insideTraffic " from the dataset
    val noOutsideTraffic = df.withColumn(" insideTraffic ",
222                             networkFilterUDF(col(ip_src.name))).filter(col("
      insideTraffic "))

224 // Drop the unneeded boolean column from the dataset
    val df = noOutsideTraffic.drop(" insideTraffic ")
226

228 // visualization code
    val port_fiveThree = df.groupBy(ip_src.name, ip_dst.name,
```

```
230                                     udp_srcport.name).agg(count(col(ip_src.name)).as(" packets_total
        "))

232 // sort a dataframe by source port from small to large
    // and the count of packets from ip sources from large to small
234 val sorted = port_fiveThree.sort(asc("udp_srcport"), desc(" packets_total "))

236 sorted.count()
    sorted.show(50)
238
    /*********************************
240 // Summary Statistics
    // This cell calculates total average packet length per unique source IP
242 // address. Data also includes number of DNS packets and total data
    // transmitted, sum of packet lengths, per unique IP source.
244 // This can be graphed as average packet length to DNS packet count or
    // by total bytes transmitted per IP to name two options.
246 *********************************/

248 // Count sent DNS packets per unique IP source.
    val packetCount = df.select("ip_src", "ip_dst"
250                            ).groupBy("ip_src"
                               ).agg(count(col("ip_dst")
252                            ).as("DNS_Packet_Count"))

254 // aggregate packet length by unique IP source.
    val aggregatePacketLen = df.select("ip_src", "udp_len"
256                                   ).groupBy("ip_src"
                                      ).agg(sum(col("udp_len")
258                                   ).as("DNS_Pkt_Len_Sum"))

260 // join the two dataframes together
    val joinDF = packetCount.join(aggregatePacketLen,Seq("ip_src"))
262
    // take packetCount and the aggregation of udp_packet length and divide to find average
264 val totalAveragePacketLengthPerUniqueSourceIP = joinDF.withColumn("average_packet_len", toInt($"
        DNS_Pkt_Len_Sum" / $"DNS_Packet_Count"))

266 // Visualization code
```

```
268 totalAveragePacketLengthPerUniqueSourceIP.count()
    totalAveragePacketLengthPerUniqueSourceIP.orderBy(col("DNS_Packet_Count").desc).show(50)
270
    /*********************************
272 // Add a Time Window to the Dataset
    // Creates a time window that is leveraged for most features
274 *********************************/

276 // use the time window function to create a time window column and add it to the dataset
    val dfWithWindows = df.withColumn(time_window, computeWindowUDF(col(frame_time_epoch.name))
        )
278
    // Frequency Distribution of Packet Sizes
280 // This feature takes unique source IP and calculates the frequency of
    // packet size ranges over a time window. The packet size ranges are
282 // calculated based on the Freedman-Diaconis rule and referred to as bins.

284 // get data ready for histogram function by rounding all UDP Packet sizes
    val dfUdpDouble = dfWithWindows.withColumn("udp_len_float", ($"udp_len" / 10)
286                             ).withColumn("udp_len_int", toInt(col("udp_len_float"))).drop("udp_len_float
        "
                                ).drop("udp_len"
288                             ).withColumn("udp_len_double", toDouble(col("udp_len_int"))).drop("
        udp_len_int")

290 // Compute the endpoints of the Histogram 'buckets' and map the rows to each 'bucket'
    val bucketDF = mapValueToBucketIndexUDF(computeBucketEndPoints(dfUdpDouble, "udp_len_double"
        ))
292
    // add the bucket column to the dataframe
294 val bktDF = dfUdpDouble.withColumn("udp_len_bucket",bucketDF(col("udp_len_double")))

296 // produces a dataframe that can be graphed to show frequency histogram of packets per bucket
    val packetLengthHistogram = bktDF.select("time_window", "udp_len_bucket", "ip_src"
298                                         ).distinct.groupBy("udp_len_bucket"
                                            ).agg(count(col("ip_src")).as("Packets_per_bucket"))
300

302 // Visualization code
    packetLengthHistogram.count
```

```
304 packetLengthHistogram.orderBy(col("udp_len_bucket").asc).show(10)

306 /*********************************
    // Packet Length Distribution and DNS Traffic Volume
308 // These features create a relationship between source IP addresses and
    // packet length. Unique IP addresses were looked at over a window of time
310 // and their packet length was aggregated and added as a column to the data set.
    // This feature is designed to help identify unique source IP addresses that
312 / /were sending high volumes of DNS traffic to a unique destination IP address.
    *********************************/
314 
    // Aggregate sums of packet lengths and counts of packets per time windows.
316 val intermediateDF = dfWithWindows.groupBy(ip_src.name, time_window).agg(
        sum(col(udp_len.name)).as("Sum_Packet_Length"),
318     count(col(udp_len.name)).as("Packet_Count")
    )
320 
    // Compute and add the average packet length per unique source ip per time window to the dataset
        .
322 val allFeaturesDF = intermediateDF.withColumn("average_packet_len", $"Sum_Packet_Length" / $"
        Packet_Count")

324 // Visualization code
    allFeaturesDF.count()
326 allFeaturesDF.orderBy(col("Packet_Count").desc).show()

328 // Summary Stats over Time Windows
    // Circadian Rhythm Statistics
330 
    // aggregate packet volume by time window.
332 val packetCountByTime = allFeaturesDF.select("time_window", "Packet_Count"
                                        ).groupBy("time_window"
334                                     ).agg(sum(col("Packet_Count")
                                        ).as("Packet_Count_Over_Time"))
336 
    // aggregate average packet length by time window
338 val avgPacketSizeByTime = allFeaturesDF.select("time_window", "average_packet_len"
                                        ).groupBy("time_window"
340                                     ).agg(avg(col("average_packet_len")
                                        ).as("Avg_Packet_Size_Per_Window"))
```

```
342
    // join the two dataframes together
344 val SumStatsOverTime = packetCountByTime.join(avgPacketSizeByTime,Seq("time_window"))

346 // total dataset packets
    val totalPackets = SumStatsOverTime.select("Packet_Count_Over_Time"
348                                            ).agg(sum(col("Packet_Count_Over_Time")
                                               ).as(" totalPacketsInDataset "))
350
    // Visualization code
352 SumStatsOverTime.count()
    SumStatsOverTime.orderBy(col("time_window").desc).show()
354  totalPackets .show()

356 /*********************************
    // Machine Learning Dataframe To CSV Conversion
358 // The transformed dataframes from above are converted to CSV files
    // and put into S3 buckets for follow-on Machine Learning
360 *********************************/

362 // write files from the distributed file system to CSV files

364 // all the columns
    allFeaturesDF . repartition (1) . write . format("com. databricks . spark . csv"
366                                       ). option ("header", "true"
                                          ). mode("overwrite"
368                                       ). save( packetFeaturesPath )

370 // Visualization Dataframe to CSV Conversion
    // The transformed dataframes from above are converted to CSV files
372 // and put into S3 buckets for follow-on visualization

374 // write files from the distributed file system to CSV files


376
    // Summary stat dataframe for whole dataset per packet
378 totalAveragePacketLengthPerUniqueSourceIP . coalesce (1) . write . format("com. databricks . spark . csv"
                                                   ). option ("header", "true"
380                                                ). mode("overwrite"
                                                   ). save(summaryStatsPath)
```

```
// histogram dataframe
packetLengthHistogram.coalesce(1).write.format("com.databricks.spark.csv"
                                            ).option("header", "true"
                                            ).mode("overwrite"
                                            ).save(packetHistPath)

SumStatsOverTime.coalesce(1).write.format("com.databricks.spark.csv"
                                            ).option("header", "true"
                                            ).mode("overwrite"
                                            ).save(TimeWindowSummaryStatsPath)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F: Machine Learning Code

```
#/**************************************************************
#* File Name: Machine Learning and Inference Pipeline using
#Batch Transform
#* Description: Trains the model on SageMaker and uses Batch
#Transform to score the data.
#* Language: Python
#* Author: LT Michael Laws (michael.laws@nps.edu)
#* Contributor: LT Greg Bunder (greg.bunder@nps.edu)
#* Thesis Advisor: Dr. Vinnie Monaco (vinnie.monaco@nps.edu)
#**************************************************************/

# Imports
# Adds necessary Amazon and Python Libraries

# Amazon Specific
import s3fs
import boto3
import botocore
import sagemaker
from sagemaker import RandomCutForest

# Common Python
import ipaddress
import sys
import io
import os
import numpy as np
import pandas as pd

# Parameter Definitions
# Configurable parameters to allow for ease of changing program functionality to include
    SageMaker and S3 Bucket Configuration

# S3 bucket parameters for Sagemaker and datasets
bucket = 'bucket' # NPS Specific Value Scrubbed
```

```
36 # ML S3 Prefixes
   datasetPrefix = 'MLpipeline/PacketFeatures/allFeatures'
38 ML_prefix = 'MLpipeline/RCF/AllFeatures'
   batch_prefix = 'MLpipeline/PacketFeaturesBatch'
40 batch_results_prefix = 'MLpipeline/PacketFeaturesBatch/inferenceResults'

42 # The location of the batch dataset
   batch_input = 's3://{}/{}/batch/'.format(bucket, batch_prefix)
44
   # The location to store the results of the batch transform job
46 batch_output = 's3://{}/{}/inferenceResults'.format(bucket, batch_prefix)

48 # Train/validate/batch split files
   train_file = 'train_data.csv'
50 test_file = 'test_data.csv'
   batch_file = 'batch_data.csv'
52
   # variable that is used split the files to remain under batch transform limits
54 row_max = 120000

56 # configure the column names for ML and Inference
   columnsForDF = ["ip_src", "time_window", "Sum_Packet_Length", "Packet_Count", "
       average_packet_len"]
58 inferColumn = ["ip_src", "time_window", "Sum_Packet_Length", "Packet_Count", "average_packet_len
       ", "score"]

60 # SageMaker Startup
   execution_role = sagemaker.get_execution_role()
62 sess = sagemaker.Session() # set up the sagemaker session to handle API interactions
   input_mode="File"
64
   # check if the SageMaker bucket exists - code from https://github.com/awslabs/amazon-sagemaker
       -examples
66 try:
       boto3.Session().client('s3').head_bucket(Bucket=bucket)
68 except botocore.exceptions.ParamValidationError as e:
       print('Hey! You either forgot to specify your S3 bucket'
70           ' or you gave your bucket an invalid name!')
   except botocore.exceptions.ClientError as e:
```

```
72      if e.response['Error']['Code'] == '403':
            print("Hey! You don't have permission to access the bucket, {}.".format(bucket))
74      elif e.response['Error']['Code'] == '404':
            print("Hey! Your bucket, {}, doesn't exist!".format(bucket))
76      else:
            raise
78  else:
        print('Training input/output will be stored in: s3://{}/{}'.format(bucket, ML_prefix))
80
    # Defines
82  #Functions Supporting operations

84  # convert ip string to integer
    def ipstring2int (row):
86      #print(row)
        try:
88          return int(ipaddress.IPv4Address(row))
        except:
90          return (0)

92  # convert ip integer to string
    def int2ipstring (int_row):
94      try:
            return ipaddress.ip_address(int_row).__str__()
96      except:
            return (0)
98
    # split and put dataframes into S3 as csv files
100 def csv_to_s3(df, filename, keyPrefix, max_rows):
        dataframes = []
102     while len(df) > max_rows:
            top = df[:max_rows]
104         dataframes.append(top)
            df = df[max_rows:]
106     else:
            dataframes.append(df)
108
        for _, frame in enumerate(dataframes):
110         fileUpload = str(_)+ filename # ex:'1batch_data.csv'
            frame.to_csv(fileUpload, index=False, header=False)
```

```
112         boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(keyPrefix, fileUpload)
    ).upload_file(fileUpload)
            #sess.upload_data(batch_file, bucket=bucket, key_prefix=keyPrefix.format(batch_prefix))
114     print('S3 location: ' + bucket + '/' + keyPrefix)

116 # get files from S3 to read into dataframes
    def get_csv_output_from_s3(bucket, prefix):
118     s3 = boto3.resource('s3')
        obj = s3.Object(bucket, '{}'.format(prefix))
120     print(obj)
        return obj.get()["Body"].read().decode('utf-8')
122
    # get a single dataframe from at least one file in a S3 bucket by passing a bucket and a prefix.
124 # functionality is different than the S3SF method and is needed for handling CSV nuances
    def pull_and_concat_from_s3(bucket, prefix):
126     fileList = sess.list_s3_files(bucket, prefix)
        df = []
128     for _, file in enumerate(fileList):
            output = get_csv_output_from_s3(bucket, file)
130         try: # deal with the "_SUCCESS" file from Scala/Spark
                df.append(pd.read_csv(io.StringIO(output), sep=",", header=None, low_memory=False))
132         except:
                continue
134         #print(file)
        df = pd.concat(df, axis=0, ignore_index=True)
136     return df

138 # return a single dataframe from a bucket/prefix combo from S3. Using S3FS keeps the datatypes
    more useable than pd.read
    def pull_and_concat_from_s3_using_S3SF(bucket, prefix):
140     fileList = sess.list_s3_files(bucket, prefix)
        df = []
142     # use S3FS to get the dataset into a dataframe
        for _, file in enumerate(fileList):
144         fs = s3fs.S3FileSystem()
            dataset = 's3://' + bucket + '/' + file
146         with fs.open(dataset, 'rb') as f:
                try: # deal with the "_SUCCESS" file from Scala/Spark
148                 df.append(pd.read_csv(f))
                except:
```

```
                continue
        print ( file )
    df = pd.concat(df, axis=0, ignore_index=True, sort=False)
    return df

# Load the Dataset into the DataFrame
# Ensure datatypes are correct

df = pull_and_concat_from_s3_using_S3SF(bucket, datasetPrefix )
df.columns = columnsForDF

# change IP addresses from objects to integers.
# For the ML model to process the data, all the datatypes in the dataframe need to be numerical.
df['ip_src']=df['ip_src'].apply( ipstring2int )

# Split and Export Data
#Split the dataset into three sets; training, testing, and validation for batch inference.
 rand_split = np.random.rand(len(df))
 train_list = rand_split < 0.8 # train on 80% of the data
 test_list = ( rand_split >= 0.8) & ( rand_split < 0.9)
 batch_list = rand_split >= 0.8 # validate/use 20% of the data

# get the training data into its own dataframe and uploaded into S3. Leave it in one file
 df_train = df[ train_list ]
 df_train.to_csv( train_file ,index=False,header=False)
 sess.upload_data( train_file , bucket=bucket, key_prefix='{}/train'.format( batch_prefix ))
 print('S3 location: ' + bucket + '/' + '{}/train'.format( batch_prefix ))

# put validation data into S3 and break it up
 df_test = df[ test_list ]
csv_to_s3( df_test , test_file , '{}/test'.format( batch_prefix ), row_max)

# break up batch infer data into chunks and upload to S3
df_batch = df[ batch_list ]
csv_to_s3(df_batch, batch_file , '{}/batch'.format( batch_prefix ), row_max)

# Train the model
#Load the values from the training data and train the model.
#https://sagemaker.readthedocs.io/en/stable/randomcutforest.html
# specify general training job information and create the rcf object
```

```
rcf = RandomCutForest(role=execution_role,
                      train_instance_count=1,
                      train_instance_type='ml.m4.xlarge',
                      data_location='s3://{}/{}/train'.format(bucket, batch_prefix),
                      output_path='s3://{}/{}/output'.format(bucket, ML_prefix),
                      num_samples_per_tree=512,
                      input_mode=input_mode,
                      num_trees=50)

# get the data into the record_set format for rcf
train_data = rcf.record_set(df_train.values, labels=None, channel='train', encrypt=False)

# train on the dataset
rcf.fit(train_data)
print('Training job name: {}'.format(rcf.latest_training_job.job_name))

# Batch Inference
#Use Sagemaker's Batch Transform functionality to rapidly score the whole dataset and store the
    results in S3
#https://sagemaker.readthedocs.io/en/stable/transformer.html

# set up the rcf_transformer as a callable object
rcf_transformer = rcf.transformer(instance_count=1, instance_type='ml.m4.xlarge', output_path=
    batch_output)

# content_type / accept and split_type / assemble_with are required to use IO joining feature
rcf_transformer.assemble_with = 'Line'
rcf_transformer.accept = 'text/csv'

# Execute the transform job on the batch dataset in the S3 bucket
rcf_transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv',
                          split_type='Line', join_source='Input')
```

APPENDIX G: Visualization Code

```
#/***************************************************************
#* File Name: Visualization Pipeline
#* Description: Takes the dataset and visualizes the information.
#* Language: Python
#* Author: LT Michael Laws (michael.laws@nps.edu)
#* Contributor: LT Greg Bunder (greg.bunder@nps.edu)
#* Thesis Advisor: Dr. Vinnie Monaco (vinnie.monaco@nps.edu)
#***************************************************************/

# Amazon Specific
import s3fs # load files from S3
import boto3 # also load files from S3

# Common Python
import sys
import io
import os
import ipaddress # convert ip address object to int
import pandas as pd # handle dataframes
import plotly.express as px # handle visualizations
import plotly.graph_objects as go

# Defines
#Functions Supporting operations

# convert ip string to integer
def ipstring2int(row):
    #print(row)
    try:
        return int(ipaddress.IPv4Address(row))
    except:
        return (0)

# convert ip integer to string
def int2ipstring(int_row):
```

```
    try:
        return ipaddress.ip_address(int_row).__str__()
    except:
        return (0)

# split and put dataframes into S3 as csv files
def csv_to_s3(df, filename, keyPrefix, max_rows):
    dataframes = []
    while len(df) > max_rows:
        top = df[:max_rows]
        dataframes.append(top)
        df = df[max_rows:]
    else:
        dataframes.append(df)
    for _, frame in enumerate(dataframes):
        fileUpload = str(_)+ filename # ex:'1 batch_data.csv'
        frame.to_csv(fileUpload, index=False, header=False)
        boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(keyPrefix, fileUpload)
    ).upload_file(fileUpload)
    print('S3 location: ' + bucket + '/' + keyPrefix)


# get files from S3 to read into dataframes
def get_csv_output_from_s3(bucket, prefix):
    s3 = boto3.resource('s3')
    obj = s3.Object(bucket, '{}'.format(prefix))
    print(obj)
    return obj.get()["Body"].read().decode('utf-8')

# get a single dataframe from at least one file in a S3 bucket by passing a bucket and a prefix.
# functionality is different than the S3SF method and is needed for handling CSV nuances
def pull_and_concat_from_s3(bucket, prefix):
    df = []
    for key in get_matching_s3_keys(bucket=bucket, prefix=prefix):
        output = get_csv_output_from_s3(bucket, key)
        try: # deal with the "_SUCCESS" file from Scala/Spark
            df.append(pd.read_csv(io.StringIO(output), sep=",", header=None, low_memory=False))
        except:
            continue
    #print(key)
```

```
    df = pd.concat(df, axis=0, ignore_index=True)
    return df

# return a single dataframe from a bucket/prefix combo from S3. Using S3FS keeps the datatypes
    more useable than pd.read
def pull_and_concat_from_s3_using_S3SF(bucket, prefix):
    df = []
    for key in get_matching_s3_keys(bucket=bucket, prefix=prefix):
        # use s3fs to get the dataset into a dataframe
        fs = s3fs.S3FileSystem()
        dataset = 's3://' + bucket + '/' + key
        with fs.open(dataset, 'rb') as f:
            try: # deal with the "_SUCCESS" file from Scala/Spark
                df.append(pd.read_csv(f))
            except:
                continue
    #print(key)
    df = pd.concat(df, axis=0, ignore_index=True, sort=False)
    return df


'''
Copyright (c) 2012-2019 Alex Chan

https://alexwlchan.net/2019/07/listing-s3-keys/
https://github.com/alexwlchan/alexwlchan.net/blob/9a80d17de47b130772bb5433592e8fffd1d18118/
    LICENSE

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

```
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

'''

def get_matching_s3_objects(bucket, prefix="", suffix=""):
    """
    Generate objects in an S3 bucket.

    :param bucket: Name of the S3 bucket.
    :param prefix: Only fetch objects whose key starts with
        this prefix (optional).
    :param suffix: Only fetch objects whose keys end with
        this suffix (optional).
    """
    s3 = boto3.client("s3")
    paginator = s3.get_paginator("list_objects_v2")

    kwargs = {'Bucket': bucket}

    # We can pass the prefix directly to the S3 API. If the user has passed
    # a tuple or list of prefixes, we go through them one by one.
    if isinstance(prefix, str):
        prefixes = (prefix, )
    else:
        prefixes = prefix

    for key_prefix in prefixes:
        kwargs["Prefix"] = key_prefix

        for page in paginator.paginate(**kwargs):
            try:
                contents = page["Contents"]
            except KeyError:
                break

            for obj in contents:
```

```
                key = obj["Key"]
154             if key.endswith(suffix):
                    yield obj
156
def get_matching_s3_keys(bucket, prefix="", suffix=""):
158     """
    Generate the keys in an S3 bucket.
160
    :param bucket: Name of the S3 bucket.
162 :param prefix: Only fetch keys that start with this prefix (optional).
    :param suffix: Only fetch keys that end with this suffix (optional).
164 """
    for obj in get_matching_s3_objects(bucket, prefix, suffix):
166     yield obj["Key"]

168 # Parameter Definitions
# Configurable parameters to allow for ease of changing program functionality
170 # to include column headers and S3 Bucket Configuration

172 # S3 bucket parameters for Sagemaker and datasets
bucket = 'bucket' # NPS Specific Value Scrubbed
174
# Visualization S3 prefixes
176 packetHistogramPrefix = 'Visualizations/packetHistogramData'
summaryStatsPrefix = 'Visualizations/summaryStatistics'
178 TimeWindowSummaryStatsPrefix = 'Visualizations/TimeWindowSummaryStats'

180 # ML results S3 Prefixes
datasetPrefix = 'MLpipeline/PacketFeatures/allFeatures'
182 ML_prefix = 'MLpipeline/RCF/AllFeatures'
batch_prefix = 'MLpipeline/PacketFeaturesBatch'
184 batch_results_prefix = 'MLpipeline/PacketFeaturesBatch/inferenceResults'

186 # The location of the batch dataset
batch_input = 's3://{}/{}/batch/'.format(bucket, batch_prefix)
188
# The location to store the results of the batch transform job
190 batch_output = 's3://{}/{}/inferenceResults'.format(bucket, batch_prefix)

192 # Train/validate/batch split files
```

```
train_file  = 'train_data.csv'
test_file   = 'test_data.csv'
batch_file  = 'batch_data.csv'

# Dataframe Columns for graphing
histogramColumn = ["Frequency Counts of DNS Packet Sizes over Time Window","Packet Counts"]
columnsForGraphALLFeatures = ["IP Source", "Time Window", "Packet Length Sum", "Packet Count", "
    Average Packet Length"]
summaryStatsColumn = ["IP Source", "Packet Count", "Packet Length Sum", "Average Packet Length"]
inferColumn = ["IP Source", "Time Window", "Packet Length Sum", "Packet Count", "Average Packet
    Length", "Anomaly Score"]

# Visualization of Entire Dataset
#General Statistics to show that the traffic is representative of normal network traffic.

# this cell takes and visualizes the histogram output from the Scala feature pipeline.
histDF = pull_and_concat_from_s3_using_S3SF(bucket, packetHistogramPrefix)
histDF.columns = histogramColumn

# data is flat > 50
histDF = histDF[histDF['Frequency Counts of DNS Packet Sizes over Time Window'] < 50]

# need to sort values by bucket
histDF = histDF.sort_values(by=['Frequency Counts of DNS Packet Sizes over Time Window'])

#This feature takes unique source IP and calculates the frequency of packet
# size ranges over a time window. The packet size ranges are calculated based on
# the Freedman-Diaconis rule and referred to as bins.

fig = px.line(histDF, x="Frequency Counts of DNS Packet Sizes over Time Window", y="Packet
    Counts")
fig.update_xaxes(ticks="inside")
fig.update_yaxes(ticks="inside", col=1)
fig.update_layout(
     title="Histogram of Packet Sizes",
     xaxis_title="Packet Size Frequency",
     yaxis_title="Counts",
    )

fig.show()
```

```
230
    # load Dataframe from S3
232 summaryStatsDF = pull_and_concat_from_s3_using_S3SF(bucket, summaryStatsPrefix)
    # add columns to dataframe
234 summaryStatsDF.columns = summaryStatsColumn
    # convert IP addresses to integers
236 summaryStatsDF['IP Source']=summaryStatsDF['IP Source'].apply( ipstring2int )
    # sort integers
238 summaryStatsDF = summaryStatsDF.sort_values(by='IP Source', ascending=False)
    # convert back to dotted decimal ip address
240 summaryStatsDF['IP Source']=summaryStatsDF['IP Source'].apply( int2ipstring )
    summaryStatsDF.info()
242
    # Total Data Transmitted
244 #This graph represents the total amount of data that was sent from
    # individual source IP addresses over the dataset .
246
    fig = px. scatter (summaryStatsDF, x="IP Source", y="Packet Length Sum")
248 fig .update_xaxes( ticks =" inside ")
    fig .update_yaxes( ticks =" inside ", col=1)
250 fig . update_layout ( title ="Packet Length Sum over IP Source ")
    fig .show()
252
    # Summary Stats over Time Windows
254 #Visualize Circadian Rhythm

256 # Pull data from S3
    TimeWindowSummaryStatsDF = pull_and_concat_from_s3_using_S3SF(bucket,
        TimeWindowSummaryStatsPrefix)
258 TimeWindowSummaryStatsDF.info()

260 # Visualize Traffic Flow over Time Windows
    data= TimeWindowSummaryStatsDF
262 fig = make_subplots(rows=2, cols=1,
                        shared_xaxes=True,
264                      vertical_spacing =0.02,
                         subplot_titles =(" ",
266                                         )#"Average Packet Length per Time Window")
                       )
268
```

```
    fig.append_trace(go.Scattergl(
270     x=data['time_window'],
        y=data['Packet_Count_Over_Time'],
272     mode='markers',
        marker_size=4,
274     name="Packet Count"
    ), row=1, col=1)
276
    fig.append_trace(go.Scattergl(
278     x=data['time_window'],
        y=data['Avg_Packet_Size_Per_Window'],
280     mode='markers',
        marker_size=4,
282     name="Avg Packet Size"
    ), row=2, col=1)
284
    #fig.update_xaxes( title_text ="Time Window", row=1, col=1)
286 fig.update_xaxes( title_text ="Time Windows", row=2, col=1)
    fig.update_yaxes( title_text ="Packet Counts", row=1, col=1)
288 fig.update_yaxes( title_text ="Average Packet Length", row=2, col=1)

290 fig.update_layout(height=600, width=1000,
                      legend_orientation ="h",
292                   showlegend = False,
                      title_text ="Packet Counts and Average Packet Length per 10 minute Time
        Window")
294 fig.show()

296 # Visualize using the features over time windows
    # Histogram of Average Packet Counts over Time Windows
298 # load Dataframe from S3
    allFeaturesDF = pull_and_concat_from_s3_using_S3SF(bucket, datasetPrefix )
300
    allFeaturesDF.info()
302 allFeaturesDF.sample(20)

304 # histfunc = ['count', 'sum', 'avg', 'min', 'max']
    # histnorm = ['', 'percent', 'probability', 'density', 'probability density']
306
    data = allFeaturesDF
```

```
# sort data by time window score
data = data.sort_values(by=['time_window']) #, ascending=False)


fig = px.histogram(data, x="time_window", y="Packet_Count", histfunc='avg')

fig.update_layout(
    title="Histogram of Average Packet Counts over Time Windows",
    xaxis_title="Time Window",
    yaxis_title="Packet Counts"
)

fig.show()


# Visualize with anomaly scores
# Get the scored data from S3 then pull the anomalous data above
# a certain threshold into a new dataframe for easier analysis.

# Pull Transformed data from S3
inferDF = pull_and_concat_from_s3(bucket, batch_results_prefix)

inferColumn = ["IP Source", "Time Window", "Packet Length Sum", "Packet Count", "Average Packet
    Length", "score"]
inferDF.columns = inferColumn
inferDF = inferDF.astype({"Average Packet Length": int, "score": int})
inferDF['IP Source']=inferDF['IP Source'].apply(int2ipstring)
# sort the data based on time. Aids in visualizing the seperate IP addresses
inferDF = inferDF.sort_values(by=['Time Window'], ascending=True)
# calculate score cuttofs for easier visualzation
score_mean = inferDF.score.mean()
score_std = inferDF.score.std()
score_cutoff = score_mean + 5*score_std


# fix columns
inferColumn = ["IP Source", "Time Window", "Packet Length Sum", "Packet Count", "Average Packet
    Length", "Anomaly Score"]
inferDF.columns = inferColumn
```

```
# Add Anomalies to the visualizations
# make a new dataframe with just data that we care about
df_anomalous = inferDF[inferDF['Anomaly Score'] > score_cutoff ]

# Anomalous Data Visualization
#We isolated the anomalous data above a certain threshold in the
# previous cell and in the next cells we visually represent
# and attempt to gain meaning from how the ML algorithm scored the information .

df_anomalous.info ()
inferDF. info ()

# Percent of data that is anomalous
#Calculate what percent of data after removing everything below the score
# cutoff will be used in the following graphs.

percentAnomalous = (len(df_anomalous) / len(inferDF))*100
percentAnomalous


# Visual representation of how the ML algorithm scored the Packet Length Sum Feature

# feature graphed over the anomaly score
data= inferDF
fig = go.Figure(data=go. Scattergl (x=data['Anomaly Score'],
                                    y=data['Packet Length Sum'],
                                    mode='markers',
                                    marker_size=4,
                                    marker_color=data['Anomaly Score'],
                                    marker_colorscale='Bluered' ,
                                    text =data['Anomaly Score'],
                                    hoverinfo=" all "))

fig . update_layout (
    title ="Packet Length Sum from IP address within each Time Window over Anomaly Scores",
    xaxis_title ="Anomaly Score",
    yaxis_title ="Packet Length Sum",
   )
```

```
fig.show()

# same graph but time window over the feature
data = inferDF

fig = go.Figure(data=go.Scattergl(x=data['Time Window'],
                                  y=data['Packet Length Sum'],
                                  mode='markers',
                                  marker_size=4,
                                  marker_color=data['Anomaly Score'],
                                  marker_colorscale='Bluered',
                                  text=data['Anomaly Score'],
                                  hoverinfo="all"))

fig.update_layout(
    title="Packet Length Sum from IP address within each Time Window colored by Anomaly Scores"
    ,
    xaxis_title="Time Window",
    yaxis_title="Packet Length Sum",
    )

fig.show()

# 3D representation of anomalous IP Source addresses over time windows
fig = px.scatter_3d(df_anomalous,
                    x='IP Source',
                    y='Time Window',
                    z='Anomaly Score',
                    color='Anomaly Score',
                    size='Packet Length Sum',
                    size_max=18,
                    symbol='Anomaly Score',
                    opacity=0.9,
                )

# tight layout
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0))
fig.update_layout(legend=dict(x=.85, y=.99))
fig.update_layout( legend_title='Score Legend')
fig.update_layout(legend={'traceorder':'reversed'})
```

```
426 fig.show()

428 # Histogram of Source IP addresses by Anomaly Score
    # To provide the best value to the watchstander and network operator,
430 # providing a summary list of IP addresses to follow up on seems to provide
    # the most easily digestable data. The below graph demonstrates that only
432 # a few IP addresses out of the original list of thousands show above the baseline.

434 # save our original dataframe
    data = df_anomalous
436
    # histfunc = ['count', 'sum', 'avg', 'min', 'max']
438 # histnorm = ['', 'percent', 'probability', 'density', 'probability density']

440 # OPTIONAL CODE FOR CHART READABILITY
    # sort data by anomaly score
442 data = data.sort_values(by=['Anomaly Score'], ascending=False)
    # do the above so we can get rid of the low scorers
444 data = data.iloc[:1300]
    # END OPTIONAL CODE
446 fig = px.histogram(data, x="IP Source", y="Anomaly Score", histnorm='probability density')

448 fig.update_layout(
        title="Normalized Probability Density of Anomaly Scores by IP Source over Time Windows",
450     xaxis_title="IP Source",
        yaxis_title="Normalized Anomaly Score"
452     )

454 fig.show()

456 # This graph is slightly different than the previous
    # as a groupby and aggregate is done over the anomaly scores first
458
    # aggregate scores by ip address over time window.
460 dfA = df_anomalous.groupby(['Time Window', 'IP Source']).agg({
            'Anomaly Score':sum    # Sum duration per group
462     }
    )
464
```

```
# flatten the dataframe from multiindex
dfA = pd.DataFrame(dfA.to_records())

# ['count', 'sum', 'avg', 'min', 'max']
# histnorm = ['', 'percent', 'probability', 'density', 'probability density']
fig = px.histogram(df1, x="IP Source", y="Anomaly Score", histnorm='probability density') #
    histfunc='avg')

fig.update_layout(
    title="Normalized Probability Density of Aggregated Anomaly Scores by IP Source over Time
    Windows",
    xaxis_title="IP Source",
    yaxis_title="Normalized Anomaly Score"
   )
fig.show()
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] J. M. Richardson, "A design for maintaining maritime superiority 2.0," Dec 2018. Available: https://news.usni.org/2018/12/17/design-maintaining-maritime-superiority-2-0

[2] U.S. Fleet Cyber Command / U.S. Tenth Fleet. U.S. Navy. [Online]. Available: https://www.public.navy.mil/fcc-c10f/Pages/home.aspx. Accessed Jan. 20, 2020.

[3] A. M. Turing, "Lecture to the London Mathematical Society on 20 February 1947," in *MD COMPUTING*, 1995, vol. 12, p. 390.

[4] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. New York, NY: Pearson Education, 2005.

[5] E. Touger. (2018, Aug). What's the Difference Between Artificial Intelligence (AI), Machine Learning, and Deep Learning? *Prowess Consulting*. [Online]. Available: https://www.prowesscorp.com/whats-the-difference-between-artificial-intelligence-ai-machine-learning-and-deep-learning/

[6] A. Ibañez. (2019, Sep). Semi-Supervised Learning... the great unknown. *Think Big*. [Online]. Available: https://business.blogthinkbig.com/semi-supervised-learning-the-great-unknown/

[7] H. Zimmermann, "OSI reference model-the ISO Model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980.

[8] M. Raza. (2018, Jun). What is the OSI Model? Network architecture in 7 layers. [Online]. Available: https://www.bmc.com/blogs/osi-model-7-layers/

[9] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: A Machine Learning Perspective*. Boca Raton, FL: CRC Press, 2013.

[10] E. R. Harold, *Java Network Programming*. Sebastopol, CA: O'Reilly Media, Inc., 2004.

[11] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver, "Practical comprehensive bounds on surreptitious communication over dns," in *Presented as Part of the 22nd USENIX Security Symposium*, 2013, pp. 17–32.

[12] A. Idris. (2018, Jul). Confusion matrix. [Online]. Available: https://medium.com/@awabmohammedomer/confusion-matrix-b504b8f8e1d1

[13] G. Al-Naymat, M. Al-Kasassbeh, and E. Al-Hawari, "Exploiting snmp-mib data to detect network anomalies using machine learning techniques," in *Proceedings of SAI Intelligent Systems Conference*, 2018, pp. 991–1004.

[14] T. Ahmed, B. Oreshkin, and M. Coates, "Machine learning approaches to network anomaly detection," in *Proceedings of the 2nd USENIX workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007, pp. 1–6.

[15] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," in *International Conference on Machine Learning*, 2016, pp. 2712–2721.

[16] M. Bartos, A. Mullapudi, and S. Troutman, "Rrcf: Implementation of the robust random cut forest algorithm for anomaly detection on streams," *Journal of Open Source Software*, vol. 4, no. 35, p. 1336, 2019.

[17] Amazon SageMaker Developer Guide. (n.d.). Amazon. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html. Accessed Feb. 17, 2020.

[18] The Scala Programming Language. (n.d.). Scala Center. [Online]. Available: https://www.scala-lang.org/. Accessed Feb. 17, 2020.

[19] Apache Spark: Unified Analytics Engine for Big Data. (n.d.). The Apache Software Foundation. [Online]. Available: https://spark.apache.org/. Accessed Feb. 17, 2020.

[20] D. Borthakur *et al.*, "Hdfs architecture guide," *Hadoop Apache Project*, vol. 53, no. 1-13, p. 2, 2008.

[21] Hadoop MapReduce. (2020). [Online]. Available: https://hadoop.apache.org/

[22] M. Yi. (2019, Sep). A complete guide to histograms. [Online]. Available: https://chartio.com/learn/charts/histogram-complete-guide/

[23] J. D. Triveri. (2019, Mar). Determining histogram bin width using the Freedman-Diaconis Rule. [Online]. Available: http://www.jtrive.com/determining-histogram-bin-width-using-the-freedman-diaconis-rule.html

[24] Random Cut Forest (RCF) Algorithm. (n.d.). Amazon. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/randomcutforest.html. Accessed Apr. 23, 2020.

[25] tshark - Dump and analyze network traffic. (n.d.). Wireshark Foundation. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html. Accessed Apr. 24, 2020.

[26] A. Coffel, "Integrated Navy Operations Support System (INOSS) Concept of Operations (CONOPS) version 0.8 - working draft," 2019.

[27] J. Robinson, "Integrated Navy Operations Support System (INOSS) framework version 0.5 - working draft," 2019.

[28] D. Arpin. (2018, Apr). Use the Amazon SageMaker local mode to train on your notebook instance. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/use-the-amazon-sagemaker-local-mode-to-train-on-your-notebook-instance/

[29] D. Ariely. (2013, Jan). Facebook. [Online]. Available: https://www.facebook.com/dan.ariely/posts/904383595868

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