# LEVERAGING MACHINE-LEARNING TO ENHANCE NETWORK SECURITY

Salazar, Daniel

Monterey, CA; Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**LEVERAGING MACHINE-LEARNING TO ENHANCE NETWORK SECURITY**

by

Daniel Salazar

June 2018

| | |
|---|---|
| Thesis Advisor: | Geoffrey G. Xie |
| Second Reader: | John D. Fulp |

**Approved for public release. Distribution is unlimited.**

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

| | | |
|---|---|---|
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** June 2018 | **3. REPORT TYPE AND DATES COVERED** Master's thesis |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** LEVERAGING MACHINE-LEARNING TO ENHANCE NETWORK SECURITY | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Daniel Salazar | |
| **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(E**S) N/A | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |

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

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

**13. ABSTRACT (maximum 200 words)**

This research examines the use of machine-learning techniques to identify malicious traffic in an emulated tactical computer network. The intent is to identify low-cost solutions based on open-source software capable of employment on computer hardware of currently fielded tactical data networks. These machine-learning techniques are investigated for application where it is prohibitive to employ bulky alternate network security measures such as security information and event management products. These methods are evaluated as a complementary solution to existing security measures, rather than as a replacement.

A test network is established with sixteen hosts emulating generation of normal baseline traffic for periods of 48 hours. One machine is infected with a botnet simulator and sends malicious traffic at four levels of intensity. The traffic flows are captured, labeled, and used as training and testing sets for four commonly used machine-learning algorithms to generate models for identifying the botnet traffic. The trained models are then tested against other flow datasets to evaluate their ability to classify malicious traffic without prior signatures. We identify the J48 Decision Tree as the strongest single algorithm across six of our seven metrics. Our work also produces a report for network administrators that is clear, easy to understand, and most importantly, provides actionable information that can drive decisions to best defend the network.

| | |
|---|---|
| **14. SUBJECT TERMS** machine-learning, botnets, network security | **15. NUMBER OF PAGES** **81** |
| | **16. PRICE CODE** |

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

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

# LEVERAGING MACHINE-LEARNING TO ENHANCE NETWORK SECURITY

Daniel Salazar
Captain, United States Marine Corps
BE, State University of New York Maritime College, 2012

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2018**

Approved by: Geoffrey G. Xie
Advisor

John D. Fulp
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

This research examines the use of machine-learning techniques to identify malicious traffic in an emulated tactical computer network. The intent is to identify low-cost solutions based on open-source software capable of employment on computer hardware of currently fielded tactical data networks. These machine-learning techniques are investigated for application where it is prohibitive to employ bulky alternate network security measures such as security information and event management products. These methods are evaluated as a complementary solution to existing security measures, rather than as a replacement.

A test network is established with sixteen hosts emulating generation of normal baseline traffic for periods of 48 hours. One machine is infected with a botnet simulator and sends malicious traffic at four levels of intensity. The traffic flows are captured, labeled, and used as training and testing sets for four commonly used machine-learning algorithms to generate models for identifying the botnet traffic. The trained models are then tested against other flow datasets to evaluate their ability to classify malicious traffic without prior signatures.  We identify the J48 Decision Tree as the strongest single algorithm across six of our seven metrics. Our work also produces a report for network administrators that is clear, easy to understand, and most importantly, provides actionable information that can drive decisions to best defend the network.

THIS PAGE INTENTIONALLY LEFT BLANK

## TABLE OF CONTENTS

# LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| **ACL** | Access Control List |
| **AI** | Artificial Intelligence |
| **ANW2** | Advanced Network Wideband Waveform |
| **ARFF** | Attribute-Relation File Format |
| **AS** | Autonomous System |
| **ATC** | Authorization to Connect |
| **ATO** | Authorization to Operate |
| **BoNeSi** | Botnet Simulator |
| **Bps** | Bytes per Second |
| **C2** | Command and Control |
| **COTS** | Commercial-off-the-Shelf |
| **CTU** | Czech Technical University |
| **DARPA** | Defense Advanced Research Projects Agency |
| **DoD** | Department of Defense (United States) |
| **DoDIN** | Department of Defense Information Network |
| **DDOS** | Distributed Denial of Service |
| **DDS-M** | Data Distribution System-Modular |
| **DNS** | Domain Naming Service |
| **EO** | Executive Order |
| **ESP** | Encapsulated Security Payload |
| **FN** | False Negative |
| **FP** | False Positive |
| **GIG** | Global Information Grid |
| **GNU** | GNU’s Not Unix |
| **GPL** | GNU Public License |
| **HACCS** | Harnessing Autonomy for Countering Cyberadversary Systems |
| **HBSS** | Host Based Security System |
| **HMSAS** | Hatch Mounted Satellite Access System |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol-Secure |

| | |
|---|---|
| **IA** | Information Assurance |
| **IAM** | Information Assurance Module |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IP** | Internet Protocol |
| **IRC** | Internet Relay Chat |
| **MAE** | Mean Absolute Error |
| **ML** | Machine-Learning |
| **NAT** | Network Address Translation |
| **TCP/IP** | Transport Control Protocol / Internet Protocol |
| **TN** | True Negative |
| **TP** | True Positive |
| **OSPF** | Open Shortest Path First |
| **PCAP** | Packet Capture |
| **PII** | Personally Identifiable Information |
| **POLP** | Principle of Least Privilege |
| **pps** | Packets per Second |
| **RFC** | Request for Comment |
| **UDP** | User Datagram Protocol |
| **UPS** | Uninterruptable Power Supply |
| **URL** | Uniform Resource Locator |
| **USMC** | United States Marine Corps |
| **WSUS** | Windows Security Update System |

# ACKNOWLEDGMENTS

First and foremost, I cannot say thank you enough to my beautiful wife, Ashley. You are incredible and have been a great support and my biggest fan. Your belief in me is truly humbling, and I hope to live up to that every day.

I am also extremely grateful for my parents, Javier and Carolyn, for all their hard work and sacrifice to get me to where I am today. To my siblings, Bryan and Michelle, a constant source of encouragement, spirit lifting, and unique perspective on life. To my in-laws, Tom and Carol, whose advice and encouragement is a true blessing.

To the multitude of professors who poured time and energy into helping this Marine begin to understand the complex and wonderful world of computer science, I am also very grateful. In particular, Dr. Xie could not have been a more encouraging, insightful, and optimistic advisor despite all the roadblocks and complications along the way of this thesis. Much credit must also be given to Professor Fulp for inspiring me to pursue network security despite its unavoidable uphill battle and difficulty.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. OVERVIEW

This research focuses on finding ways to improve network security in the Department of Defense (DoD) through open-source software compatible with currently fielded hardware. We establish an emulated test network to provide a baseline of normal user traffic. We then introduce a simulated botnet infection to launch a Distributed Denial of Service (DDoS) attack on another controlled network. Subsequently, we compare the efficacy and suitability of various machine-learning (ML) algorithms to see which yields the best results at identifying the malicious behavior from normal baseline traffic.

## B. MOTIVATION

Network security is a fundamental necessity for successful military operations in the 21st century. Virtually all pertinent information for the operational and strategic levels of warfighting will transit computer networks. More and more, computer networks are being pushed to the "tactical edge" as well. With the proliferation of systems like the AN/PRC-117G radio and Hatch Mounted Satellite Access System (HMSAS) and technologies like Advanced Network Wideband Waveform (ANW2), leveraging the strengths of computer networks at the tactical level is becoming part of our overall plan to stay ahead of our adversaries. But for all the advantages of tactical level computer networks, they also present many vectors for attack. This research is motivated by a desire to find low cost solutions to enhance network security that can be implemented immediately.

## C. SCOPE

Tactical computer networks are characterized by their small size, short duration of operation, and expeditionary locations. Generally, a tactical network will support as few as one user to as many as two hundred. They are designed to be maintained anywhere from a few hours to a few months. Typically, they are established in "field conditions" with temporary structures, a temporary power grid, and temporary access to the global information grid (GIG). These characteristics contrast sharply with those of garrison networks, which are established in permanent structures with reliable power and can support thousands of users for an indefinite time period. Another significant distinguisher between tactical and garrison networks is their resources. Because a garrison network is larger, it has a more well-funded defense in depth approach to network security. This includes both trained professionals and sophisticated equipment such as security incident and event managers (SIEM). Meanwhile, because of the quantity of tactical networks, it is cost-prohibitive to deploy expensive sophisticated security solutions to each one. Also, any hardware solutions are further complicated by the need to ruggedize equipment for transport my military vessel (land, sea, or air) and deployment in austere conditions (heat, sand, moisture). Because of these factors, tactical networks do not have nearly the robustness for network security that their garrison counterparts do. This particular research will focus on network security for these smaller, more expeditionary tactical networks.

Even though the networks may be smaller at the tactical level, they are susceptible to all the same vulnerabilities of a larger network. They can be interrupted by a DDoS attack or infiltrated via spam, spear phishing, or other social engineering tactics. The malicious activity at the center of this research is botnets. A more detailed explanation of botnets can be found in Chapter II.

The scope of this research is on identifying computers in a tactical network that have been compromised as part of a botnet. Methods for identifying these compromised machines are restricted to only use commodity hardware and open source software. The intent of this work is to find solutions that can be implemented under current fiscal restrictions and that do not require the purchase of expensive hardware for each tactical network.

## D. OBJECTIVES

The primary objective of this research is to find a low-cost solution to identify compromised machines on a tactical network by analyzing network traffic with machine-learning algorithms. The term “low-cost,” as used here, means no additional hardware or software purchases are necessary to implement the solution on current tactical networks.

## E. ASSUMPTIONS

Although the term "tactical network" is rather broad, for the purposes of this research, it is assumed that such a network has modest firewall capability in the form of access control lists (ACLs) and possibly a pre-programmed commercial firewall. It is also assumed that there are modest computing resources available for conducting network traffic analysis in search of compromised machines.

## F. APPROACH

The overall approach is to conduct tests of various machine-learning algorithms on simulated network traffic. First, a test network based on a small USMC tactical network is established. This test network generates benign normal traffic as our emulated users perform normal tasks over time. After this, a botnet malware simulator is deployed on the test network from a single machine, and the traffic of this "infected" machine is labeled as such. With this labeled traffic, a number of machine-learning algorithms and voting methods are tested to find which ones are best suited to identify botnet traffic over a range of metrics.

## G. BENEFITS OF RESEARCH

A low-cost method to identify compromised machines would enhance overall DoD network security without taking resources from other priorities. While it is not possible to address every possible threat with any single solution, adding a machine-learning based approach could tighten our network security posture and ultimately benefit any DoD network that employs this method.

## H. ORGANIZATION

Chapter II provides background on USMC tactical network architecture, botnet traffic, and machine-learning algorithms. In Chapter III, the design of the experiments is laid out. Chapter IV covers the results of these experiments. Finally, Chapter V provides our conclusion and suggests possible avenues for future research to build off the methodology, data, and lessons learned from this research.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. BACKGROUND

This research starts by establishing a knowledge baseline of current USMC tactical computer networks. This includes current system equipment and the security features in place to protect these smaller, short-term networks. Then a discussion on botnets looks at what they are, why they are so prevalent, and dives a little deeper into common command and control (C2) configurations. Finally, the topic of machine-learning is explored. We look at the fundamentals of what makes machine-learning software as well as recent advancements that have brought this technology into the mainstream.

## A. OVERVIEW OF USMC TACTICAL NETWORKS

Since this research is focused on improving network security of USMC tactical networks, it is important to note the current equipment and procedures in place.

### 1. Standard Equipment and Architecture

It is important to understand the quantity and capability of the equipment currently fielded by the USMC in its tactical networks because this research is focused on providing solutions that can be implemented with this pre-existing infrastructure. There are dozens of commercial grade solutions on the market, but they require the procurement of specialized hardware. For the DoD, any equipment that will connect to the Department of Defense Information Network (DoDIN) must go through additional security screening and scrutiny before receiving an authorization to connect and authorization to operate (ATC/ATO). So, in the interest of finding solutions that can be implemented with minimal delay, it is preferable to work within the currently approved hardware.

USMC tactical networks are based around the Data Distribution System-Modular (DDS-M). The DDS-M is a set of modules, each with different capabilities, that can be used in a variety of combinations to create a network tailored to the unit's needs on a given mission. Each of these modules is centered on a commercial-off-the-shelf (COTS) hardware component combined with an uninterruptable power supply (UPS) and mounted in a ruggedized case for use in field environments. For user computers, the USMC fields a number of approved COTS laptops. Brands include Panasonic, Lenovo, and Dell, and the technical specifications are commensurate with mid-level commodity laptops available to the general public. Based on these current operating conditions, this research is conducted on similarly capable hardware. Details of this hardware and how it is used for these experiments can be found in Chapter III.

### 2. Current Network Security Measures and Limitations

For tactical networks, the USMC employs several defensive measures. In order to connect a tactical network to live internet services, the network must utilize a firewall. The Information Assurance Module (IAM) is a ruggedized Fortinet firewall. Also, the network must use Host Based Security System (HBSS), the McAfee designed enterprise software to protect end devices. Additionally, network administrators use the Windows Security Update Services (WSUS) to patch machines on the network.

Beyond technology and equipment, there is a significant burden of labor for network administrators. These service members require current entry-level network security certification, such as CompTIA’s Network+ and Security+ courses. Once trained, they are responsible for maintaining the defensive posture of the network. This includes port restrictions, enforcing password policy requirements, and maintaining best practices for Microsoft’s Active Directory, such as configuring settings in accordance with the principle of least privilege (POLP). However, while these individuals are trained and execute adherence to security policy, they do not actively look for threats to the network. In larger USMC tactical networks, an Information Assurance (IA) Marine can be attached to supplement the unit’s cyber security. These IA Marines have increased training and are responsible for monitoring the network through a collection of tools. The most prominent of these tools are the firewalls and the network vulnerability scanners.

A major problem in both the technologies and procedures that we currently employ is their reliance on signature-based threat detection. If a known virus is attached to an email, the firewall will flag it when it compares the hash to its database of known malware. This alerts an IA Marine to investigate and take the necessary corrective actions. If a user’s machine is infected with a known virus, HBSS will notice and send a message in a similar manner. The problem is that with today's rate of new malware creation, a signature-based solution is not enough. According to Symantec, there were 357,019,453 new malware variants in 2016 and 669,947,865 in 2017 [1]. Not only do signature-based solutions fail to address the volume and velocity of new malware, but malicious actors are always developing new ways to disguise their malware and evade these tools.

One current gap in our network defense is when new malware (signature not yet catalogued) or self-hiding malware (signature changes or location of code hidden from AV tools) infects a user's machine and executes without the user noticing (i.e. not ransomware that alerts a user their data has been encrypted). One common "family" of malware that can fall into this category are botnets. This research focuses on this particular type of malware due to its growing use and the threat it poses to DoD networks. An overview of botnets follows in Section 2.B.

In the cases described above with botnets or other malware, a behavior-based detection method is needed. Extensive research has been done and is currently being conducted on behavior-based methods, and some significant progress has been made. One area in particular has been machine-learning. A number of research initiatives have explored how to apply machine-learning to the detection of anomalous network traffic [2]. This research focuses on using open source machine-learning software to identify malicious behavior on a tactical network. An overview of machine-learning follows in Section 2.C.

## B. OVERVIEW OF BOTNETS

Most simply stated, a botnet is a group of computers controlled by one person. Each computer in a botnet is called a "bot" because it is running software that automates its functionality (i.e., does not need human interaction to execute). It is called a bot*net* because of the coordination between all the distributed bots making it a network. While technically this could describe many legitimate—non-malicious—networks managed by network administrators (running WSUS involves one human controlling a network of machines running automated updating scripts), the term botnet has come to be understood as a malicious construct. In fact, some go so far as to say botnets “are responsible for a vast majority of the spam on the Internet today” [3].

### 1. Defense and Security Considerations

The DoD is not immune from this threat. Indeed, as more and more DoD systems rely on the Internet, our systems are at greater risk than ever. While over a decade ago, in 2005 a botnet was discovered operating on DoD computers [4] and resulted in a stand-down to wipe the infected machines. More recently, in January 2018 a joint report to the President by the US Secretaries of Commerce and Homeland Security was released titled “A Report to the President on Enhancing the Resilience of the Internet and Communications Ecosystem against Botnets and Other Automated, Distributed Threats.” [5]. This report was a product of the president’s issuing of Executive Order (EO) 13800 “Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure” [6]. This EO and report both indicate how seriously the botnet threat is to national security, as well as our economy as a whole. In 2017, the Defense Advanced Research Projects Agency (DARPA) began soliciting for solutions to the botnet threat (among others) in its initiative “Harnessing Autonomy for Countering Cyberadversary Systems (HACCS)” [7]. The project hopes to leverage current advances in ML and artificial intelligence (AI) to automate the process of defending networks against botnets and other threats.

One of the reasons that botnets have garnered such a high threat level is that they are extremely flexible. In fact, it may be more appropriate to refer to botnets as a theory or principle rather than a technology. This is based on the fact that botnets can be created in a variety of ways, controlled in a variety of ways, and deliver a variety of adverse effects in a variety of ways. No single technology or algorithm defines botnets. They have achieved their current level of proliferation thanks to the abundance of network resources (bandwidth, end devices, et al.) that has been on the rise for decades. Because the DoD has been increasingly reliant on this same increase in network resources, the threat of botnets does not appear to be dwindling any time soon.

### 2. Common Command and Control Methods

If there is one defining feature of botnets, it is their command and control. Indeed, it may be the only thing separating a botnet from just viruses and worms doing damage on the Internet. The ability to control a multitude of compromised machines in a matter of seconds is what gives botnets their teeth. The controller of a botnet is often referred to as a "bot herder." In order to issue commands and receive information from their bots, a communication channel, or C2 channel, must be used. Early botnets trended toward use of the Internet Relay Chat (IRC) protocol as their C2 channel. Fortunately for network security professionals, many current networks do not use IRC so it is a simple matter to disable this port at their firewall. But to no one's surprise, the C2 methods of bot herders have only been limited by their imaginations. Because nearly all Internet connected device rely on Domain Naming Service (DNS), Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol-Secure (HTTPS), these have been favorites of bot herders to hide their C2 traffic. This is done by pointing their bots to the Uniform Resource Locator (URL) of a website they either control or are injecting with their malicious commands. In fact, Symantec reports that in 2017 approximately 1 in every 88 (~1.1%) of all URLs on the Internet were for botnet activity, primarily C2 [1].

## C. OVERVIEW OF MACHINE-LEARNING

While sometimes used synonymously with artificial intelligence, machine-learning is not exactly the same. Both terms are prolific in the realm of "big data" analytics and have received a great deal of attention due to recent advances. Artificial intelligence is the broader term covering all aspects of simulating intelligence, that is decision making, learning, self-correcting, through the use of computer systems. Machine-learning is one critical component of artificial intelligence wherein a computer system can adapt and improve on its own without human correction. The rise of machine-learning is perhaps most significantly a paradigm shift in fundamental programming philosophy. Before this area began to gain traction, it was always assumed that the human would have to apply critical thinking to problems that had never been seen before and then adjust code on a computer as necessary to handle the new scenario. In short, the human would write a program telling the computer not just what to do, but how to do it. No matter how complicated, theoretically any action that the computer took could be seen as originally coming from the human programmer. For a computer to play checkers, a programmer could go through each possible state of the playing board and decide for the computer what move it should make in that exact scenario, and then write each of these "if-then" rules into the code.

Machine-learning is a fundamental change to this paradigm that was sparked by the desire for computers to solve problems and handle situations that were far too large computationally for this method. A classic example is the game of chess, which has far more possible board states and moves than checkers. To program a computer to successfully play chess, no programmer in the world could possibly include actions to take in every state of chess, estimated to be $10^{45}$. So, in order to handle these increasingly complex problems, the idea of machines learning came about. The idea was that humans solve problems, like a chess game, not by knowing every single possible scenario, but by understanding the rules and objectives and figuring out strategies based on these parameters. So, the fundamental idea of machine-learning is to mimic this in software. Instead of coding scenarios, the programmer defines the rules of the game and provides the computer with quantitatively defined incentives. In chess for example, the taking of an opponent's pawn might have an incentive value of 2, while the taking of an opponent's queen might have an incentive value of 20. However, there may also be a negative incentive for exposing the computer's *own* queen of say -30. So, when the computer is calculating its move, it weighs all incentives for a particular state and chooses the path with the highest incentive reward.

In ML applications, each data point is referred to as an instance or case; for purposes of this research we will use the term instance. Each instance has a set of values associated with it, known as features. For example, in a ML application trying to predict the weather, it might be "fed" millions of instances of past meteorological data. Each instance would likely be the data from a specific hour at a specific location, and its features could be anything from temperature to humidity to barometric pressure.

A key aspect to machine-learning methods is the idea of a training set and a testing set. The training set is a collection of data in which the answer is provided. Continuing the example of weather prediction, the training set would be a set of instances with all the features of a location at a certain time, with a label for what weather condition occurred next. In the case of a machine vision algorithm, where a computer determines what an image represents, it is a collection of images labeled with the correct interpretation. The computer is fed this training set and told that it is the ground truth, that these answers are in fact correct. From this training set, the algorithm can adjust its parameters to best fit with that training set. For example, in a computer vision application if 99% of the pictures in the training set with the label "lake" had over 75% of the pixels in the blue spectrum, it could set a parameter that told it to be 99% confident that an image with over 75% blue pixels was a lake. From there, the testing set is very similar in the fact that it is a collection of data with correct labels. However, when the testing set is provided to the computer, the labels are removed. After it decides its answers for each case, it then goes back and compares its answers to the actual answers. By doing this, it can determine how effective it was at learning based on the training set. This is also a chance for programmers to evaluate how well suited a particular algorithm is for a particular use case.

### 1. Algorithms

The various ways to incentivize a program and to search for the best path in each state has led to the invention of a number of varied solutions. Since machine-learning is a field of computer programming, not a program itself, there is a plethora of algorithms that attempt to best handle these challenges. What is interesting is that certain algorithms have achieved impressive results in certain specific domains. There is no one machine-learning algorithm that is best suited for all applications. The Institute of Electrical and Electronics Engineers (IEEE) held the International Conference on Data Mining in December 2006. At this conference, the top ten algorithms used in data mining were identified and published in a survey paper [8]. Contributors to the conference came from universities and businesses across the globe. Three of the four algorithms investigated in this research are in this paper: Naïve Bayes, J48 Decision Tree, and AdaBoost. The fourth algorithm we look into is Logistic Regression. Logistic Regression is a well-established ML algorithm but does not appear in many efforts to classify network traffic data [9]. It is included as a reference point for comparison.

### *a. J48 Decision Tree*

One such family of machine-learning algorithms is called decision trees. These end up resembling something akin to flowcharts in the business world whereby each decision leads to another branch in which new factors are weighed and another decision is reached. These tend to be best suited for application in which there are a small number of attributes for the data. Too many attributes lead to an unmanageable number of branches and the searching becomes too computationally exhaustive to be effective.

Ross Quinlan developed the C4.5 algorithm as a decision tree that improved upon his previous ID3 algorithm. It has since been quite popular as a supervised classifier, even being called "a landmark decision tree program that is probably the machine-learning workhorse most widely used in practice to date" [10]. The Weka software used by this research and described later in this section features an open-source Java implementation of the C4.5 algorithm called a J48 decision tree.

### *b. Naïve Bayes*

Another approach to machine-learning was developed by applying Bayesian probabilities. In Bayesian machine-learning, we apply our knowledge of prior probabilities to give weight to certain features. This allows us to apply some of what humans have already learned into the algorithm, so it is not starting blind. For instance, in the case of a machine-learning program to assist a doctor in diagnosing lung cancer it can be encoded that the prior probability of a patient who smokes to have lung cancer is 10%, while the prior probability of a patient who is 10 years old to have lung cancer is 0.1%. Now, when the computer is looking at all the patient’s data, it can be more confident in a diagnosis of lung cancer if the patient smoked than in the patient was 10 years old. While many times these weights can be programmed in directly, it is very common to provide a Bayesian model with a training set and let it determine what the prior probabilities are based on that data. For this research, we evaluate the Naïve Bayes algorithm because it assumes strong independence of the features and has been studied extensively in other machine-learning efforts to classify network traffic [9].

#### *c. Logistic Regression*

Within the family of regression statistics, the Logistic Regression model has been adapted for use in ML classifiers. Typically, Logistic Regression yields probabilities of a certain instance being a certain value. For adaptation as a binary ML classifier, cutoff values are assigned to the probabilities and if the value is greater than the cutoff, it is classified as a positive. Since Logistic Regression is a common and well-established method, it is used as one of our algorithms for comparison. While it is not commonly studied for use in network data classification, it is included as a reference point.

#### *d. AdaBoost—An Ensemble Learning Technique*

Ensemble learning is the practice of combining several ML algorithms to achieve better performance. One subset of ensemble learning is known as boosting. Boosting is the process of creating a model using an algorithm, and then building a subsequent model that attempts to correct the errors of the first. Typically, this is most effective when used with weak classifiers. The AdaBoost method is one such technique that uses one level decision trees, also known as decision stumps, as the weak classifier. It is best suited for binary classification and has been used extensively in ML, which is why it was selected for inclusion in this research.

#### *e. Ensemble Learning via Voting*

Another technique for ensemble learning is voting. The premise is to take the results of several algorithms and then take a vote to determine the final classification. In this way, the strengths of one algorithm may be helpful to cover the weaknesses of another. This research uses three different voting schemes to evaluate possible result improvement. The first is the one vote minimum scheme which simply states that if any of the algorithms classifies an instance as a positive, then the result is to classify it as a positive. On the opposite end of the spectrum, we also evaluate the all or none voting scheme. In this method, the final result is only classified as positive if all the algorithms classified it as positive. The middle ground is using the simple majority voting method. As the name implies, if the majority of voting algorithms classify an instance as a positive, then the final result is classified as a positive.

### 2. Measures of Effectiveness

There are two dominant metrics used to measure the accuracy of machine-learning algorithms: precision and recall. Each of these use counts of true positives, false positives, true negatives, and false negatives. These can best be understood in a simple case of a classifier determining whether an image is of a dog (in class D) or not of a dog (not in class D).

- True positive (TP): a case that has been correctly classified as belonging to class D. I.e. ML algorithm called it a dog and it was actually a dog, so it was right.
- A true negative (TN): a case that has been correctly classified as not belonging to class D. I.e. ML algorithm said it was not a dog and it was actually a cat, so it was right.
- A false positive (FP): a case that has been incorrectly classified as belonging to class X. I.e. ML algorithm said it was a dog and it was actually a cat, so it was wrong.
- False negative (FN): a case that has been incorrectly classified as not belonging to class X. I.e. ML algorithm said it was not a dog and it was actually a dog, so it was wrong.

These four values are often reported in a binary confusion matrix. The matrix provides a quick glance at the results of the algorithm and can provide insight into its suitability for a particular use case.

Table 1. Binary Confusion Matrix Example

| Classified as: | Class D | Not Class D |
|---|---|---|
| Class D | TP | FP |
| Not class D | FN | TN |

Calculating all the cases as one of these four results allows the calculation of precision (P) and recall (R) as follows.

$$P = \frac{TP}{(TP+FP)} \tag{1}$$

$$R = \frac{TP}{(TP+FN)} \tag{2}$$

Precision can be thought of as the *usefulness* of the results, while recall can be thought of as the *completeness* of the results. That is to say precision tells us how many of the positive results yielded are actually positive, while recall tells how many positive results we found out of all the actual positive results.

It is common to plot the recall versus precision results of a classification ML algorithm to view its performance characteristics while altering the threshold. This helps decision makers decide where the appropriate threshold should be set for their specific application of these algorithms. A more risk-averse scenario would prefer a lower threshold. This favors more false positives but catches more true positives. On the contrary, a use case that can afford to take more risk may increase the threshold in order to maximize availability of its system. To capture both precision and recall in a single value, the F-measure is another common metric used:

$$F_{measure} = 2 \times \frac{(P \times R)}{(P+R)} \tag{3}$$

Additionally, we will use both the mean absolute error (MAE) and the root mean squared (RMS) error to gain insight to the performance of our algorithms. The following formulas are used where n is the total number of instances being evaluated. For our purposes with binary classification, each instance will have a value of either 1 or 2 for the respective class. This means an accurate prediction would have an error of 0 for that instance, and an inaccurate prediction means an error of 1 for that instance.

$$MAE = \frac{\sum|predicted-actual|}{n} \tag{4}$$

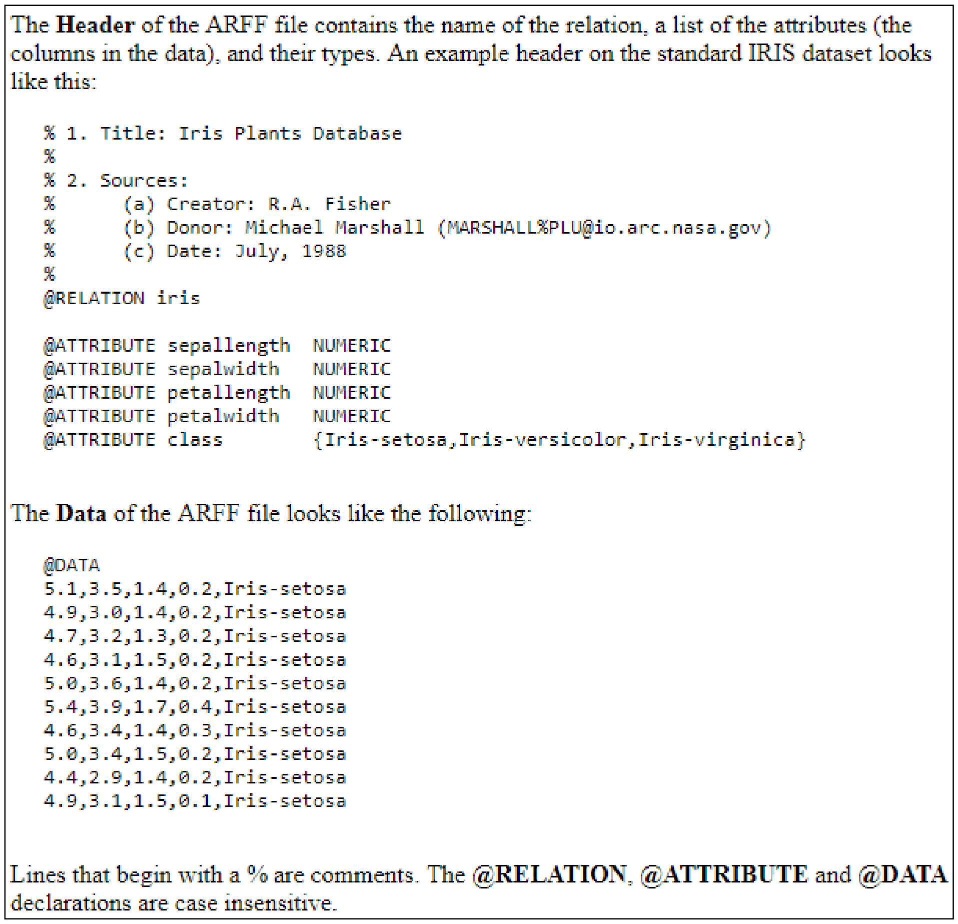
$$RMS = \sum \frac{\sqrt{(predicted-actual)^2}}{n} \tag{5}$$

For this research, we consider an instance to be positive if it is a botnet flow. That is to say that unless a classifier determines an instance to be suspected of being botnet traffic, it will classify the instance as a negative result. In this manner, a true positive is a successfully identified botnet flow while a true negative is a correctly classified normal flow. The truly interesting cases are the false positives and false negatives. When a normal flow is incorrectly classified as a botnet flow, this is a false positive. Depending on the use case, a network administrator may prefer an algorithm that provides more false positives just to be sure they are catching everything. However, it could also be possible to have an algorithm that provides so many false positives that it is impossible to investigate each. Other algorithms will favor false negatives, whereby a botnet flow gets incorrectly identified as normal. This may or may not be acceptable, depending on the network being evaluated. By looking at all these measurements of effectiveness, we seek to find the best algorithms for application in different networks.

### 3. Weka

The Machine-Learning Group at the University of Waikato in New Zealand has developed and maintained the Weka project. According to their site [11], "Weka is a collection of machine-learning algorithms for data mining tasks." This software is open-source and issued under the GNU General Public License. The hallmark of the Weka software is the ability to provide users a vast number of machine-learning algorithms to use on their own datasets. The convenience of being able to perform these operations from a single platform has made Weka a favorite among researchers for being able to quickly compare the accuracy of various algorithms. In addition to the software itself, a burgeoning wealth of supporting documentation is growing each day as more people adapt the software for their needs. As of the time of this writing, the latest stable release is 3.8.0.

While Weka does support many complex algorithms, it does require a very specific input file format that was developed by the same group at the University of Waikato. It is a text-based file named Attribute-Relation File Format (ARFF) with the appropriate file extension "arff" to designate as such.  For most users, the burden of using Weka is getting data into the ARFF format for processing by the software. There are two main sections to an ARFF file, the header, which defines all the data relationships and data types, and the data itself. Figure 1 is an example ARFF file from the developers' site.


The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

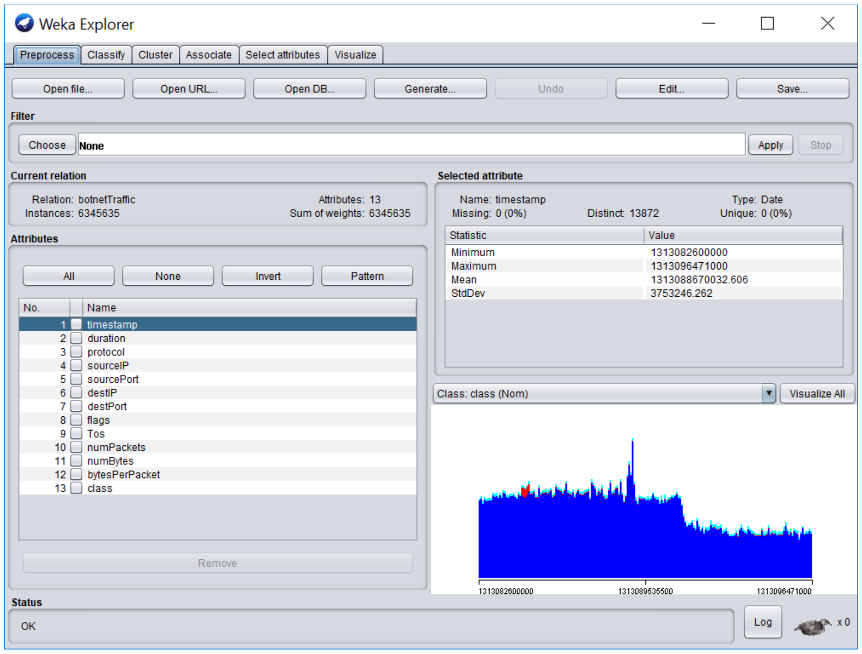
The **Data** of the ARFF file looks like the following:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Lines that begin with a % are comments. The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive.

Figure 1. Example ARFF File with Notations. Source: [11].

The Weka software starts with pre-processing the data provided. In this part, Weka loads an ARFF file and does error checking as well as initial summary statistics. Figure 2 shows the graphical user interface (GUI) in this stage. From here, a user can filter out certain data attributes, see both numeric and graphical summaries of each attribute, and even invert attributes. While Weka does support a number of data types for attributes, not every algorithm supports every data type. For example, many algorithms including Naïve Bayes and Logistic Regression do not allow for the 'date time' data type. In addition, Naïve Bayes does not allow for strings either. This makes the preprocessing step very important for tailoring a dataset to be used by a specific algorithm.



Filter: None

Current relation

Relation: botnetTraffic
Instances: 6345635
Attributes: 13
Sum of weights: 6345635

Selected attribute

Name: timestamp
Missing: 0 (0%)
Distinct: 13872
Type: Date
Unique: 0 (0%)

| Statistic | Value |
|---|---|
| Minimum | 1313082600000 |
| Maximum | 1313096471000 |
| Mean | 1313088670032.606 |
| StdDev | 3753246.262 |

Attributes

| No. | Name |
|---|---|
| 1 | timestamp |
| 2 | duration |
| 3 | protocol |
| 4 | sourceIP |
| 5 | sourcePort |
| 6 | destIP |
| 7 | destPort |
| 8 | flags |
| 9 | Tos |
| 10 | numPackets |
| 11 | numBytes |
| 12 | bytesPerPacket |
| 13 | class |

Class: class (Nom)

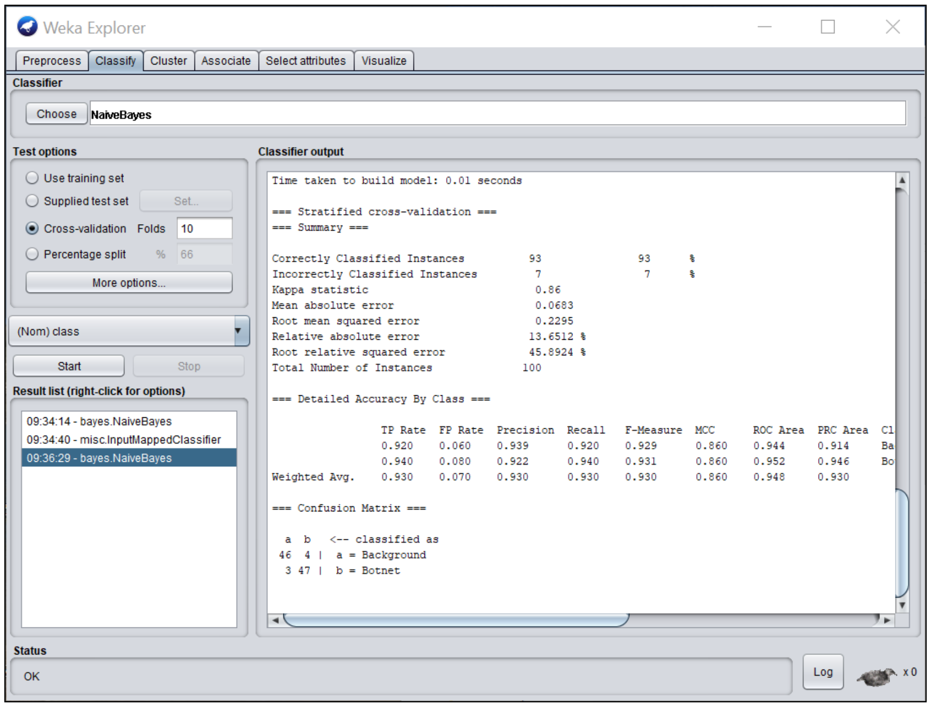
1313082600000 1313089535500 1313096471000

Status

OK

Figure 2. Weka’s GUI for Pre-processing Data

After preprocessing, Weka can move into several modes for analysis. These includes classifying, clustering, and associating. For purposes of this research, we will focus on classifying. Once in the classify tab, the user can select from the many algorithms available. Only the algorithms that allow the data types in the currently loaded dataset will be available for selection. Once a selection has been made, the user can select how to handle the separation of training data with testing data. Weka allows for the use of a training set, a testing set, a cross-validation method, and a percentage split. For cross validation, the user can select how many folds they would like. For percentage split, the user can decide the exact value as well. After running the classifying algorithm, the Weka GUI provides very valuable information regarding the results, as seen in Figure 3. This includes mean absolute error, precision and recall of each class, as well as a confusion matrix. Each of these provides valuable insight regarding the level of success with the chosen algorithm on the user’s data.



NaiveBayes

- Use training set
- Supplied test set
- [✓ Cross-validation] Folds 10
- Percentage split % 66

(Nom) class

09:34:14 - bayes.NaiveBayes
09:34:40 - misc.InputMappedClassifier
09:36:29 - bayes.NaiveBayes

```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          93               93      %
Incorrectly Classified Instances         7                7      %
Kappa statistic                          0.86
Mean absolute error                      0.0683
Root mean squared error                  0.2295
Relative absolute error                 13.6512 %
Root relative squared error             45.8924 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Cl
                 0.920    0.060    0.939      0.920    0.929      0.860    0.944     0.914     Ba
                 0.940    0.080    0.922      0.940    0.931      0.860    0.952     0.946     Bo
Weighted Avg.    0.930    0.070    0.930      0.930    0.930      0.860    0.948     0.930

=== Confusion Matrix ===

  a  b   <-- classified as
 46  4 |  a = Background
  3 47 |  b = Botnet
```

OK

Figure 3. Example of Model Summary Output in Weka's Explorer

THIS PAGE INTENTIONALLY LEFT BLANK

# III. EXPERIMENTAL DESIGN

This chapter explains the overall design of the experimentation conducted. The design process consists of three phases. In phase 0 we conduct preparations that include analyzing traffic obtained from a USMC test network and traffic generated by another university to study botnet behavior. These results help shape the following phases. In phase 1 we create a test network to generate both benign and malicious traffic. This traffic is used to build ML models that can be used to classify botnet traffic from normal traffic. In phase 2 we analyze these ML models' abilities to classify botnet traffic on additional datasets. This includes a look at the metrics that will be our measures of effectiveness to objectively determine the effectiveness of each ML model being explored.

## A. PHASE 0: PRELIMINARY ANALYSIS

We start with a phase 0 that informs and shapes the design of other elements of our experiments. In this phase we analyze traffic obtained from a test network at the Marine Corps Tactical Systems Support Agency (MCTSSA). Additionally, we use Weka to analyze pre-labeled NetFlow data with botnet traffic generated by another research group from the Czech Technical University (CTU). This analysis helps inform our selection of an optimized feature set for our experiments.

### 1. Analysis of USMC Test Network Traffic

Packet capture (PCAP) data was provided by the Marine Corps Tactical Systems Support Activity (MCTSSA) in support of this research. The PCAP files are from an evaluation conducted in 2017. It must be noted that this data provided was not from a 'live' operational unit. It was from an evaluation environment, where traffic was simulated. This also means no Personally Identifiable Information (PII) was contained in this data.

This data was first converted to NetFlow format with a Python script and then analyzed for characteristics. The NetFlow format was chosen because it is widely adopted and extensively researched, with a variety of tools available for parsing and analyzing. Also, due to the limitations of tactical networks, the NetFlow format is preferable because of the drastic reduction in size for traffic. This size reduction is important because it reduces the resources needed to collect, store, and analyze this data. In addition, because this research focuses on network-behavior-based detection methods, the NetFlow format captures the characteristics most associated with the behavior of traffic while omitting the fields that are the focus of signature-based methods, namely the packet data itself [17].

The intent of analyzing the USMC test network traffic was to better design our own test network. However, a few obstacles proved this effort untenable. First, of the 18 PCAP files given, there was great disparity among the protocols and size distribution of packets observed. Second, despite multiple attempts, the contacts who provided the data were not able to identify from where on the network each PCAP file was taken. Because of this, we were not able to do an equivalent comparison to the characteristics associated with the location of our network tap.

### 2. Machine-Learning on Pre-labeled NetFlow Dataset

In 2014, researchers at CTU created a large dataset of network traffic for use in research comparing botnet detection methods [12]. Their data is publicly available and divided into 13 separate scenarios, each with its own botnet activity mixed in. These are collectively referenced as CTU-13. Each scenario is assigned a number from 42 to 55, so the first scenario is labeled CTU-42, the second scenario is CTU-43, and so on. In each scenario's flow level data, they labeled each flow as either "Background," "Legitimate," or "Botnet." Using this flow data, we were able to run some preliminary analysis of algorithms in Weka to determine which features were the most useful. Table 2 shows the results of this analysis using the second scenario dataset with the Naïve Bayes algorithm. Of note, the date-time data type is not supported by Weka's implementation of the Naïve Bayes Algorithm. Also, of note, the IP addresses were converted to numeric values. Source and destination IP addresses are grouped into one feature category because they are either included or excluded together. The same is true of source and destination ports.

The first run of the algorithm included all possible features, resulting in only a 10.6379% success rate. The algorithm was run several more times with each run having a different feature excluded. At the end of these runs, it was found that only three features—

the source and destination IP address, number of packets, and number of bytes—resulted in better performance when they were excluded (highlighted in green in the table). The next runs involved excluding two out of three of these features in each combination possible, and then finally excluding all three. This final run of removing all three troublesome features resulted in the best performance: 89.6373%.

The results here told us that using the IP addresses as numeric values was very detrimental to the results. This is because the numeric relationship between IP addresses is not significant, and actually misleading. In these algorithms, numeric values are compared as relative to one another. This means an IP address of 1.2.3.4 is seen as closer to 2.3.4.5 by several orders of magnitude compared to 192.168.0.1. However, because IP addresses do not share commonality by proximity in numeric value, but rather by subnet relationship and neighbor relationships, this is a flawed way to approach the problem. Fortunately, our research seeks to find discoveries of malicious flows that can be implemented in multiple networks, so the models must be time and IP address agnostic. For our experiments, we never use the datetime value or IP addresses as features.

With this revelation handled, we looked to the other detrimental features. The number of packets and the number of bytes were features that reduced our percentage of correct results. Because of this, we design our later experiments to run each algorithm with two sets of features: maximum and optimized. The maximum feature set includes all NetFlow values except datetime and IP addresses while the optimized feature set further remove the packets and bytes features.

Table 2. Comparison of Feature Selection with Naïve Bayes Algorithm Applied to the CTU-43 (Scenario 2) Dataset

| Features and data type | | | | | | | | | | Accuracy Result |
|---|---|---|---|---|---|---|---|---|---|---|
| timestamp[1] | duration | protocol | sourceIP & destIP | sourcePort & destPort | flags | TOS | num Packets | num Bytes | bytes Per Packet | |
| date time | numeric | nominal | numeric | numeric | nominal | numeric | numeric | numeric | numeric | |
| ■ | | | | | | | | | | 10.6379% |
| ■ | ■ | | | | | | | | | 10.1664% |
| ■ | | ■ | | | | | | | | 3.7334% |
| ■ | | | ■ | | | | | | | 82.7454% |
| ■ | | | | ■ | | | | | | 1.5266% |
| ■ | | | | | ■ | | | | | 3.7131% |
| ■ | | | | | | ■ | | | | 10.6379% |
| ■ | | | | | | | ■ | | | 12.2258% |
| ■ | | | | | | | | ■ | | 12.8863% |
| ■ | | | | | | | | | ■ | 9.6658% |
| ■ | | | ■ | | | | ■ | | | 84.6897% |
| ■ | | | ■ | | | | | ■ | | 85.2810% |
| ■ | | | ■ | | | | ■ | ■ | | 89.6373% |

Legend

■ Shaded cell indicates feature removed

Note 1: The date/time data structure is not supported by Weka's implementation of Naïve Bayes.

## B. PHASE 1: MACHINE-LEARNING MODEL GENERATION

Once the preliminary analysis of phase 0 was complete, the main experiments of this research could commence. Figure 4 shows the flow of the experiments in phase 1 from the generation of traffic by our emulation network, through the ML model creation.

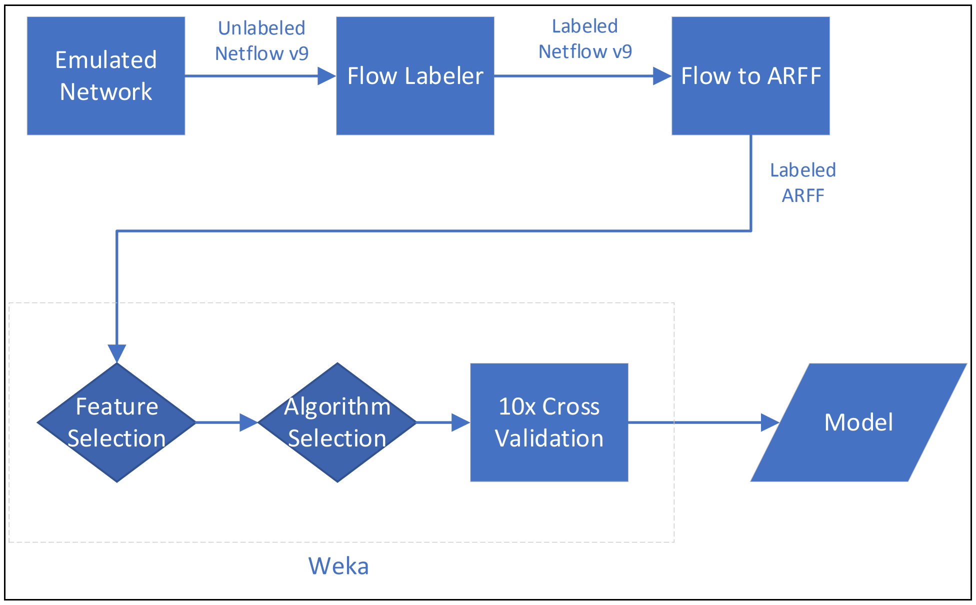


Figure 4. Phase 1 Process Flow: Using an Emulated Network to Generate an ML Model

To start, the emulated network generates normal baseline traffic. Details of the test network design are in Section C of this chapter. We then inject simulated botnet traffic from one of the computers in this network to mix in with the normal traffic. All the traffic is captured in NetFlow v9 and saved. This NetFlow data is then passed through our flow labeler. The accuracy of this step is key to producing an accurate model. We use attributes of the simulated botnet traffic that we control (timestamps, IP addresses) in each experiment to tailor this labeler. Once labeled, the flows are converted into the ARFF file format to be handled by Weka. In Weka, we adjust which features are included in the analysis in an effort to select the ones that provide the most useful data. As mentioned earlier, this feature selection is heavily influenced by our study of the CTU-13 dataset. Once the features are selected, we try a number of ML algorithms available in Weka.

For each of these algorithms, we run the labeled flow data through with 10x cross-validation for the training and testing phases. The cross-validation method is chosen because of its widely accepted status in statistical analysis. In cross-validation, the entire dataset is divided into the training set and the testing set. The algorithm is then run and statistics of performance are maintained. The entire dataset is then split once again into training and testing data, and the algorithm run again for k iterations, also called folds. This happens for any given number of folds (in our case 10), and the division is conducted in such a way that each instance serves as testing data exactly one time. This helps reduce the common problem in ML of "overfitting." Overfitting is when the algorithm tailors itself so closely to the training and testing data provide that it is not useful for any other data. Since this would defeat the purpose of building a model, we use 10x cross-validation to get the best model possible. Figure 5 provides a visual depiction of this process. The model created from this step will be used for the next phase of the experiment.

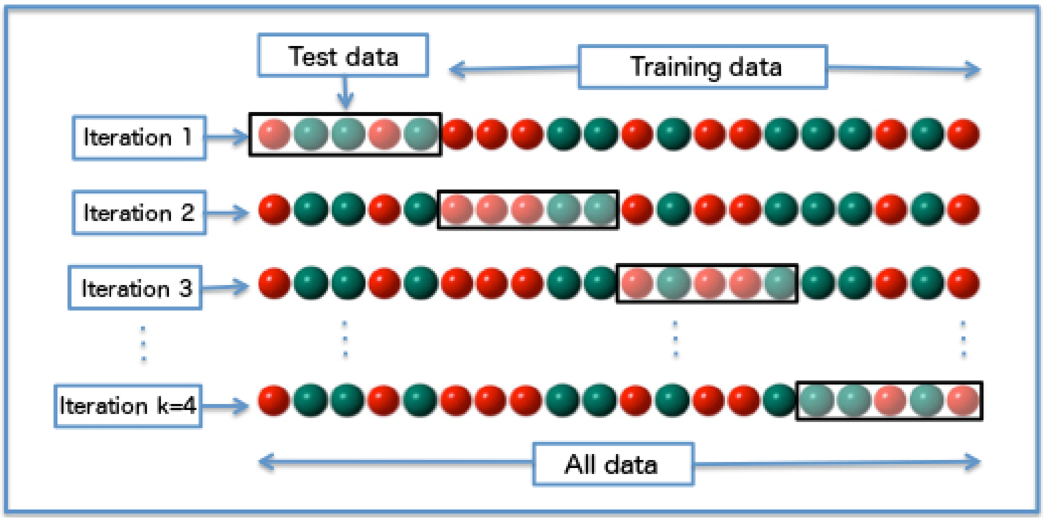


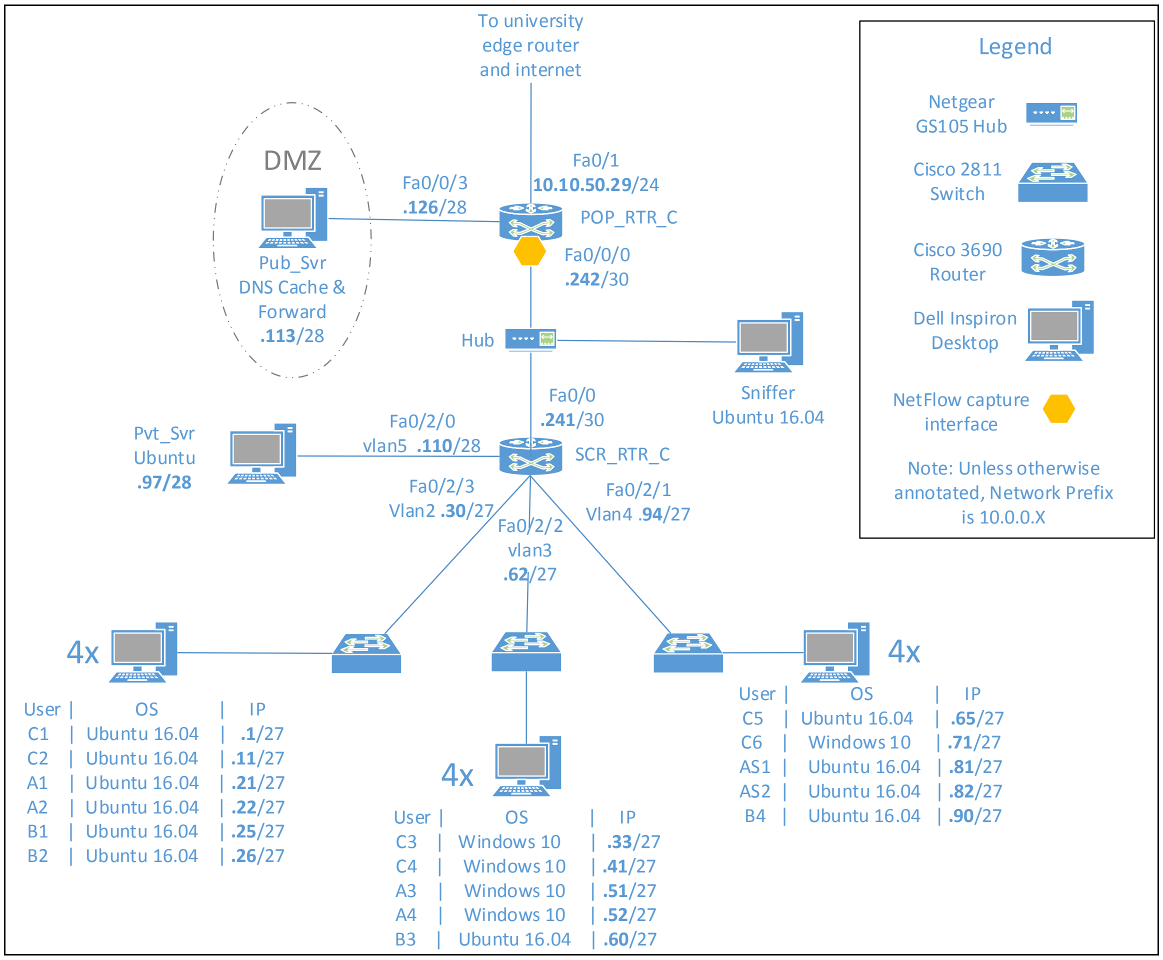
Figure 5. Representation of k-folds Cross-Validation with k=4. Source: [13].

### 1. Test Network Design

The test network was designed and configured with commodity hardware and open source software. Any computational limitations by using this method are intentional due to the desired application of these methods to current USMC tactical networks where commodity hardware is prevalent.

#### *a. Network Configuration*

The test network is designed based on the basic construct of a USMC tactical network from the researchers’ experience. The network features a point of presence (POP) and screening (SCR) router configuration. A public facing web server is located in the demilitarized zone (DMZ) off of the POP router which provides DNS caching and forwarding. A private server is hosted on a subnet off of the SCR router. Three client subnets are also connected to the SCR router, each with four client machines. All addressing uses private IP space per request for comment (RFC) 1918, and the overloaded network address translation (NAT) is handled by the POP router. The network employs Open Shortest Path First (OSPF) for routing within the Autonomous System (AS). Figure 6 is a diagram for this test network.



| User | OS | IP |
|---|---|---|
| C1 | Ubuntu 16.04 | .1/27 |
| C2 | Ubuntu 16.04 | .11/27 |
| A1 | Ubuntu 16.04 | .21/27 |
| A2 | Ubuntu 16.04 | .22/27 |
| B1 | Ubuntu 16.04 | .25/27 |
| B2 | Ubuntu 16.04 | .26/27 |

| User | OS | IP |
|---|---|---|
| C3 | Windows 10 | .33/27 |
| C4 | Windows 10 | .41/27 |
| A3 | Windows 10 | .51/27 |
| A4 | Windows 10 | .52/27 |
| B3 | Ubuntu 16.04 | .60/27 |

| User | OS | IP |
|---|---|---|
| C5 | Ubuntu 16.04 | .65/27 |
| C6 | Windows 10 | .71/27 |
| AS1 | Ubuntu 16.04 | .81/27 |
| AS2 | Ubuntu 16.04 | .82/27 |
| B4 | Ubuntu 16.04 | .90/27 |

Figure 6. Test Network Diagram

### 2. Host Configuration

Each client machine had a freshly installed operating system (OS). Nearly half of the client machines were running Ubuntu 16.04, while the remaining machines ran Windows 10. In order to emulate user traffic, each client machine ran its own Python script that automated tasks a normal user would likely have done. These tasks include reading and sending email, as well as searching the web and downloading files. Since this research focuses on network traffic, the automation scripts did not include other common user tasks like reading and writing to files on the local machine. Each user had a profile that dictated the percentage of their network activity that was email based versus browser based. This was meant to correlate to role disparity usually found in operational units. For instance, it is common for management roles to be more email intensive in their network traffic and for other users to be more focused on browsing for information. Each user also had a business factor variable that determined how many tasks they would complete on any given shift. Pseudo-random numbers were used to simulate realistic discrepancies in time to complete tasks and time between tasks in a human user's activity. The scripts were also tailored to simulate a typical user's diurnal patterns of life, working a set amount of time per day and taking breaks.

#### *a. Monitoring*

A network tap is set up on the link between the SCR and POP routers. This consists of a hub (Netgear GS105) with another user machine connected as a sniffer. This sniffer machine used tcpdump to capture PCAP of all traffic going into and out of the network. To record the flow level data, the SCR router was configured to export Cisco NetFlow version 9 flow records for both the ingress and egress direction on its internal interface (fa0/0). The internal interface was chosen because of the overloaded NAT being utilized. Flow records from the external interface would all show the single public IP address with port numbers assigned. There would be no efficient way to reconcile these addresses to their source addresses because the router maintains its own reference tables to handle these mappings, which change constantly. By exporting flows from the internal interface, we maintain the integrity of each flow mapping to the actual IP address of the host responsible.

### 3. Normal Traffic Generation

For the majority of each test, the user machines were busy emulating normal tasks to generate normal traffic. Each user had one of three assigned workday shifts starting at either 0800, 1600, or 0000 and lasting for eight hours. Tasks were spread throughout each user's workday, with some user's being busier than others and therefore having more tasks to complete. Some users were designed to be more email intensive while others were more browser intensive, again representing different roles for individuals in a tactical network. Efforts were taken to ensure a realistic spread of internet traffic generated. This included searching for current news to ensure that users were not always accessing cached sites and including attachments of varying sizes in emails to create far less predictable characteristics for the normal baseline traffic. All of these parameters were designed and based on the researchers' experience with tactical networks and the tasks USMC users performed on these networks.

### 4. Botnet Traffic Injection

In order to test the ML methods in this research, much consideration went into how best to generate botnet traffic. It was deemed outside the scope of this research to test "live" malware that could propagate without our knowledge. So the decision was made to explore simulators already developed. There are many commercially available tools for the penetration testing market. Many of these are robust and have more functionality than we needed, as well as being expensive.

For this research, the software chosen is the aptly named Botnet Simulator, also known as BoNeSi. It is an open-source project under the Gnu Public License (GPL) and according to its readme file, BoNeSi "is a Tool to simulate Botnet Traffic in a testbed environment on the wire. It is designed to study the effect of DDoS attacks." The tool is developed and maintained by Markus-Go on GitHub [14]. BoNeSi was chosen because it is an easy to use command line utility which provides the proper options to tailor the tool to our research. It is designed for simulating DDoS attacks and has parameters to define the target and rate of packets transmitted. It is also the only current simulator that includes functionality for TCP traffic in addition to UDP traffic. This allows us to be more confident in our results because we can control the botnet traffic generated to see how the ML methods handle both protocols. Other possible botnet activity such as sending spam or executing click fraud are not examined in the scope of this research.

From the test network, we loaded the BoNeSi software on one of the user machines (User_5). At specified times and within specified parameters we used BoNeSi to DDoS an address in a second test network. Normally the strength of BoNeSi is its ability to simulate large numbers of random IP addresses on the target machine. However, for our purposes we limit it to a single IP address, the actual host's IP address, and throttle back the rate of packets being sent. The idea is that a very noisy bot sending massive amounts of traffic in a burst to a target would be easily identifiable by existing network security measures. The much harder case to find is a single bot sending a low rate of packets as part of a much larger botnet's DDoS campaign. This is why we constrained BoNeSi in this fashion.

We conducted four experiments, each utilizing a different intensity of simulated botnet traffic. With BoNeSi this is accomplished by altering the throughput parameter to adjust the number of packets sent per second. We also send half of the malicious traffic over TCP and half over UDP. At the end of this step, we have four collections of NetFlow data, each containing varying intensity levels of simulated botnet traffic to be used for our machine-learning efforts.

### 5. Machine-Learning

In the machine-learning step, we use optimizations discovered in our phase 0 analysis of the CTU-13 dataset. First, we must pre-process the data generated by our emulation network, and then we can generate models in Weka for each combination of botnet traffic intensity and feature set.

#### *a. Pre-processing*

At this point, we had both user (packet) and session (flow) level data of both benign and malicious traffic. In order to use this data for our ML applications, we had to reliably label the data for training and testing the various algorithms. This labeling was the first step to pre-processing the data for Weka. The second part of the pre-processing involved the conversion of the flow data into the ARFF format needed by Weka. This was accomplished with a script to identify all flows to and from the target machine. In our research environment, we could ensure all botnet flows were accurately labeled because we controlled all the variables. In an actual implementation, this labeling for training would need to be done in a similar way to ensure the models are trained on accurate training sets.

#### *b. Training and Testing*

Once we have a labeled training set, we maintain continuity across experiments by always utilizing the 10-fold cross-validation method to train our models. This efficiently uses our entire dataset to train and test the model ten times. At this step, Weka also reports the time taken to build each fold of the model which we use as a secondary metric by which to evaluate these algorithms. At the end of this step, we have a model for each algorithm with which we can test against other datasets in phase 2.

## C. PHASE 2: MODEL VALIDATION

For this phase, we utilize the NetFlow data from several of our iterations before the labeling step. We provide the unlabeled version of the data to Weka and load the various models built in phase 1. For each model, we run the respective algorithm and receive an output that classifies each instance with a confidence level. From this output, we run it through a program to correlate the instance number with the IP addresses. This step is key because actual network administrators do not need to know which flows are suspected as malicious, but rather which machine on their network is causing these malicious flows. The output of this correlator is a final report that indicates to the network administrator the percentage to which it believes each machine on the network is behaving normally or abnormally. Figure 7 shows the flow for this phase of research.

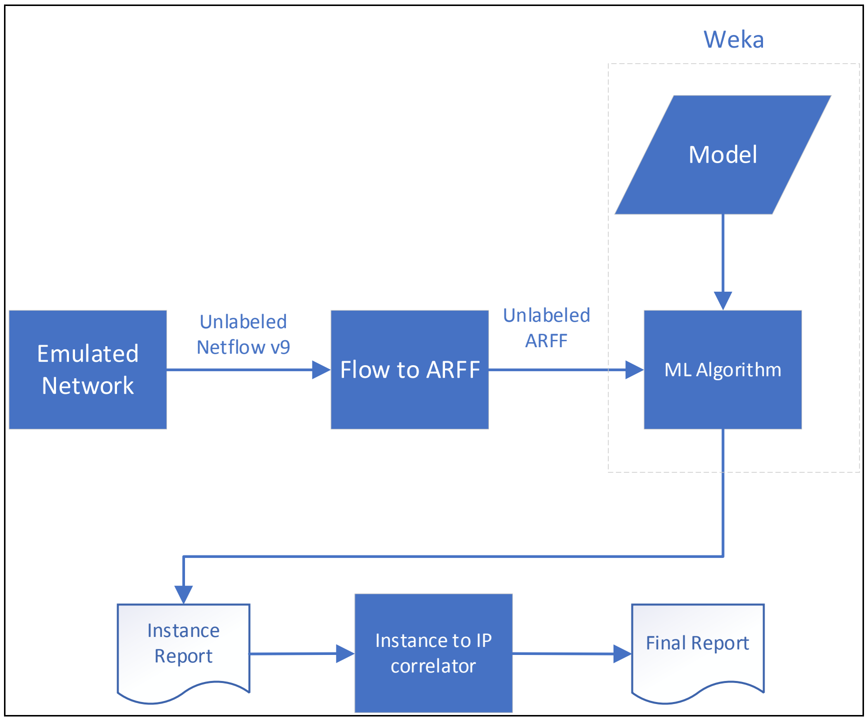


Figure 7. Phase 2 Process Flow: Validation of ML Models

The correlator is a key step in making this process useful to human operators. By aggregating the instance by IP address, the program then gives a final ranking of its confidence that each machine is behaving normally or is suspect of malware. For instance, if 100 flows for a particular source IP address are all in the range of 10-30% likely to be malicious, this may not warrant further investigation. On the other hand, if a particular source IP address has 90 flows that are <15% likely to be malicious, but 10 flows that are 80% likely to be malicious this is a much stronger indicator that the machine is compromised. This step is key in making the entire application of ML to network security usable for integration by a network administrator.

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. RESULTS

This chapter starts with an overview of the testing parameters and general observations. Then it explores each of the four algorithms individually and the six configurations for voting between those four models.

## A. TRAFFIC INTENSITY

For each iteration, the 16 user computers of the test network ran for 48 hours. At the 46th hour, the computer designated as the infected bot began sending its DDoS traffic with the BoNeSi software. The bot was allowed to send traffic for approximately 90 minutes. The four iterations conducted are hereby labeled by the relative intensity of this simulated botnet traffic. The lowest of these intensities involved the sending of only 1 TCP packet and 1 UDP packet per second to the target victim IP address. Because each of these packets is set to 32 bytes, the result was approximately 64 bytes per second (Bps) of botnet traffic. In each case, the botnet traffic was composed of 50% TCP traffic and 50% UDP traffic. Each of the parameters is outlined in Table 3.

Table 3. Botnet Traffic Intensity of Each Experiment

| **Botnet Traffic Intensity** | **Packets per Second (pps)** | **Packet Size (Bytes)** | **Bytes per Second (Bps)** | **Total Number of Flows** | **Number of Botnet Flows** | **Percent of Flows Designated Botnet** |
|---|---|---|---|---|---|---|
| **Low** | 2 | 32 | 64 | 1,670,906 | 8,374 | 0.50% |
| **Moderate** | 10 | 32 | 320 | 2,420,624 | 61,190 | 2.53% |
| **High** | 100 | 32 | 3200 | 2,139,085 | 779,676 | 36.45% |
| **Ultra-High** | 1000 | 32 | 32000 | 10,415,109 | 8,945,746 | 85.89% |

## B. TIME CONSIDERATIONS

As explained in Chapter III, the models were built using 10-fold cross validation. Weka reports the time taken to build the model for the first fold of this process. With this as a metric, it can be seen that there was a significant range of time taken to build each model based on the algorithm selected. Table 4 shows a comparison of these times. The Naïve Bayes algorithm comes out as the clear winner, taking only an average of 8.25 seconds to build a model for each fold. After ten folds, this results in 1.38 minutes. This is substantial because the second fastest algorithm, the J48 Decision Tree, is still an order of magnitude slower at 87.73 seconds per fold, or 14.6 minutes (for ten folds). The slowest algorithm was Logistic Regression, coming in at 514.56 seconds per fold and totaling 85.8 minutes (for ten folds).

Table 4. Time Comparison for Building of ML Models

| **Botnet Traffic Intensity** | **Low** | | **Moderate** | | **High** | | |
|---|---|---|---|---|---|---|---|
| **Feature Set** | **Max** | **Opt** | **Max** | **Opt** | **Max** | **Opt** | **Average** |
| **Algorithm** | **Time per Fold to Build Model (sec)** | | | | | | |
| **Logistic Regression** | 515.61 | 101.46 | 913.14 | 540.14 | 759.63 | 257.37 | 514.56 |
| **J48 Decision Tree** | 82.88 | 42.85 | 92.61 | 63.59 | 124.36 | 120.06 | 87.73 |
| **Naïve Bayes** | 10.7 | 6.16 | 7.87 | 7.53 | 11.34 | 5.9 | 8.25 |
| **AdaBoost** | 184.86 | 156.89 | 379.85 | 346.28 | 243.77 | 162.53 | 245.70 |

## C. FEATURE SET COMPARISON

For every ML application, the choice of features to include in the dataset is key [15]. In Chapter III Section 2.A. it was discovered that for the CTU-13 dataset, the removal of the IP addresses, number of packets, and number of bytes helped improve the accuracy of the results. Since this research seeks applications for ML models that are network and time agnostic, the timestamp and IP addresses are not included in any of the data sets used to build the ML models. However, we did seek to discover whether the removal of the number of packets and number of bytes would help in our data as it did with the CTU-13 dataset. We refer to the set of all features as the "Maximum" feature set, and the set of features with packets and bytes removed as the "Optimized" feature set.

By optimizing the feature set, we observed some interesting results. As to be expected, reducing the feature set sped up the model generation time for all algorithms. Table 5 shows an example of the speed improvements for the low botnet traffic intensity dataset.

Table 5. Time Improvements for Feature Set Optimization of Low Botnet Traffic Intensity Models

| **Time to Build Each Fold (sec)** | | | | |
|---|---|---|---|---|
| | **Maximum Features** | **Optimized Features** | **Time Reduction** | **% Speed-up** |
| **Logistic Regression** | 515.61 | 101.46 | 414.15 | 408.19% |
| **J48 Decision Tree** | 82.88 | 42.85 | 40.03 | 93.42% |
| **Naïve Bayes** | 10.7 | 6.16 | 4.54 | 73.70% |
| **AdaBoost** | 184.86 | 156.89 | 27.97 | 17.83% |

More surprising was how optimizing the feature set had no significant impact on the results. For Naïve Bayes, this optimization did improve the percent of correctly classified instances by 4.49% in the low botnet traffic intensity dataset. Averaging across the low, moderate, and high botnet traffic intensity datasets, the improvement was 3.32%, as seen in Table 6. This goes hand in hand with an improvement of the root mean square error of 7.97% averaged across the iterations, as seen in Table 7. However, it appears that this optimization was not a positive factor for all algorithms. The J48 decision tree was only marginally improved by 0.0002% for correct classification, and 1.46% for root mean squared error. On the other hand, both Logistic Regression and AdaBoost actually saw drastic reductions in their accuracy. AdaBoost suffered the most, with a -0.9255% change in accuracy and -43.54% change in root mean squared error.

Table 6. Improvements to Classification Accuracy by Optimizing Feature Set

| Improvement for Correctly Classified Instances (%) | | | | |
|---|---|---|---|---|
| **Botnet Traffic Intensity** | **Low** | **Moderate** | **High** | **Average** |
| **Logistic Regression** | -0.0120% | -0.0012% | -0.6219% | -0.2117% |
| **J48 Decision Tree** | 0.0005% | 0.0002% | -0.0001% | 0.0002% |
| **Naïve Bayes** | 4.4870% | 2.4137% | 3.2160% | 3.3722% |
| **AdaBoost** | -0.3868% | -1.8702% | -0.5194% | -0.9255% |

Table 7. Improvements to Root Mean Squared Error by Optimizing Feature Set

| Improvement for Root Mean Squared Error | | | | |
|---|---|---|---|---|
| **Botnet Traffic Intensity** | **Low** | **Moderate** | **High** | **Average** |
| **Logistic Regression** | -10.12% | -0.46% | -60.61% | -23.73% |
| **J48 Decision Tree** | 5.97% | 1.85% | -3.45% | 1.46% |
| **Naïve Bayes** | 9.37% | 4.85% | 9.67% | 7.97% |
| **AdaBoost** | -52.26% | -62.29% | -16.08% | -43.54% |

These changes to performance must be weighed against the respective change in time required to build the model. Table 8 shows the average improvement for build time of each fold across the low, moderate, and high botnet traffic intensity datasets.

Table 8. Model Generation Time Improvement by Optimizing Feature Set

| Improvement for Time to Build Each Fold (sec) | | | | |
|---|---|---|---|---|
| **Botnet Traffic Intensity** | **Low** | **Moderate** | **High** | **Average** |
| **Logistic Regression** | 408.19% | 69.06% | 195.15% | 224.13% |
| **J48 Decision Tree** | 93.42% | 45.64% | 3.58% | 47.55% |
| **Naïve Bayes** | 73.70% | 4.52% | 92.20% | 56.81% |
| **AdaBoost** | 17.83% | 9.69% | 49.98% | 25.84% |

## D. GENERAL OBSERVATIONS

One of the most useful abilities of ML models is their ability to be trained on a particular dataset and then applied to others. For this research, we took each model generated on its particular botnet traffic intensity level and applied it to each other botnet traffic intensity level. The resulting tables can be found in full in Appendix A.

It must be noted that while an ultra-high botnet traffic intensity dataset was collected and analyzed, the results were severely limited due to the computing capability available. For model generation, only the Naïve Bayes and Ada Boost algorithms could be used due to memory constraints. Even this was only achieved by allocating a 128GB heap size to Weka. The computation took several hours due to the intensive memory swapping that occurred from using so much storage space not typically available in memory for a heap. Also interesting was the fact that optimizing the feature set for AdaBoost actually resulted in the inability of Weka to complete the model generation. This could have been due to the particular garbage collection routine of the machine being used, or perhaps even to the fact that Weka saves the removed features somewhere in memory to allow an "undo" functionality. Furthermore, when models were tested against the ultra-high intensity dataset, the resulting files were considerably larger due to the sheer number of flows (over 10.4 million). This led to an inability to load multiple results files in memory to evaluate the voting methods on this dataset. Because this research is focused on solutions that can be implemented on commodity hardware, further calculations by more capable computing resources were not attempted.

## E. INDIVIDUAL ALGORITHMS

Each of the four algorithms tested had varying levels of performance. We compared the seven metrics that could be averaged for all test set sizes, training set sizes, and feature sets for all ten algorithms/voting methods. The results can be seen in Table 9. For each metric (category) we highlight the top three and bottom three performers.

From this overall comparison, it is evident that J48 was consistently the most successful across almost all metrics. Even the second and third best performing methods were majority voting that included J48 as one of the members. On the other hand, the one vote minimum method performed the worst in six of the seven metrics. Interestingly, it performed the best in the seventh metric: recall. This is due to the fact that recall is a measure of how many of the total positives did we actually identify. Because the one vote minimum takes all positive predictions from all the algorithms, it makes sense then that it would identify the most positives and therefore have the best recall.

Table 9. Comparison of Algorithms and Voting Methods Averaged Across All Tests

| Average Scores Across All Testing, Training, and Feature Sets | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Voting Method | None | | | | One Vote Min | All Or None | Majority | | | |
| ML Algorithm(s) | Log | J48 | NB | Ada | All | All | Log & J48 & NB | Log & J48 & Ada | Log & NB & Ada | J48 & NB & Ada |
| Correctly Classified Instances (%) | 0.9830239 | 0.9984798 | 0.8184657 | 0.8960573 | 0.774457 | 0.956249 | 0.9969721 | 0.9933512 | 0.9898472 | 0.9953267 |
| Incorrectly Classified Instances (%) | 0.0169761 | 0.0015202 | 0.1815343 | 0.1039427 | 0.225543 | 0.043751 | 0.0030279 | 0.0066488 | 0.0101528 | 0.0046733 |
| Mean absolute error | 0.0169792 | 0.0015167 | 0.1815333 | 0.1039458 | 0.2255444 | 0.04375 | 0.0030222 | 0.00665 | 0.0101556 | 0.0046667 |
| Root mean squared error | 0.1039417 | 0.0214458 | 0.4087292 | 0.2219042 | 0.4721556 | 0.1314833 | 0.0466556 | 0.0631722 | 0.0888889 | 0.0618333 |
| Precision | 0.9185292 | 0.9940125 | 0.43855 | 0.6912583 | 0.2663722 | 0.8245889 | 0.8928833 | 0.9243611 | 0.8248667 | 0.8483222 |
| Recall | 0.9520292 | 0.9970208 | 0.9936792 | 0.6801625 | 0.9996722 | 0.6714278 | 0.9966778 | 0.9603278 | 0.9582722 | 0.9946167 |
| F-Measure | 0.9241208 | 0.9954708 | 0.4966208 | 0.6513875 | 0.3395667 | 0.7147056 | 0.9309111 | 0.9334611 | 0.8627056 | 0.8987611 |

| Legend |
|---|
| Best Result in Category |
| 2nd Best in Category |
| 3rd Best in Category |
| 3rd Worst in Category |
| 2nd Worst in Category |
| Worst Result in Category |

### 1. Naïve Bayes

The Naïve Bayes algorithm performed the worst of all the individual algorithms, and the second worst among all methods in six of the seven metrics as seen in Figure 8. It consistently achieved correct classification rates in the 70–80% range when the other algorithms achieved >98% when tested on the low intensity dataset. In fact, it never achieved greater than 85% for correct classification in any permutation of variables studied. For comparison, the other three algorithms never achieved less than 90% correct classification rate. These poor results have to be counted along the one area it surpassed the other three algorithms in: time to build the model. If time is truly important, the Naïve Bayes model was built an order of magnitude faster than the other models as mentioned earlier. Unfortunately, for a ten-fold increase in speed, it provides a 72.7% accuracy compared to J48's 99.9% as seen in the low intensity trained models with maximum features tested on the same low intensity dataset. If time is truly important, such as in large scale and high bandwidth applications, perhaps this is an acceptable trade off.

A look at the confusion matrices shows that Naïve Bayes suffers from an abundance of false positive predictions. This problem carries over into the one vote minimum voting method. Because Naïve Bayes classifies so many flows as botnets, the one vote minimum also incorrectly classifies all those flows. Figure 8 shows the effect of this massive disparity by comparing the false negatives on a logarithmic scale.

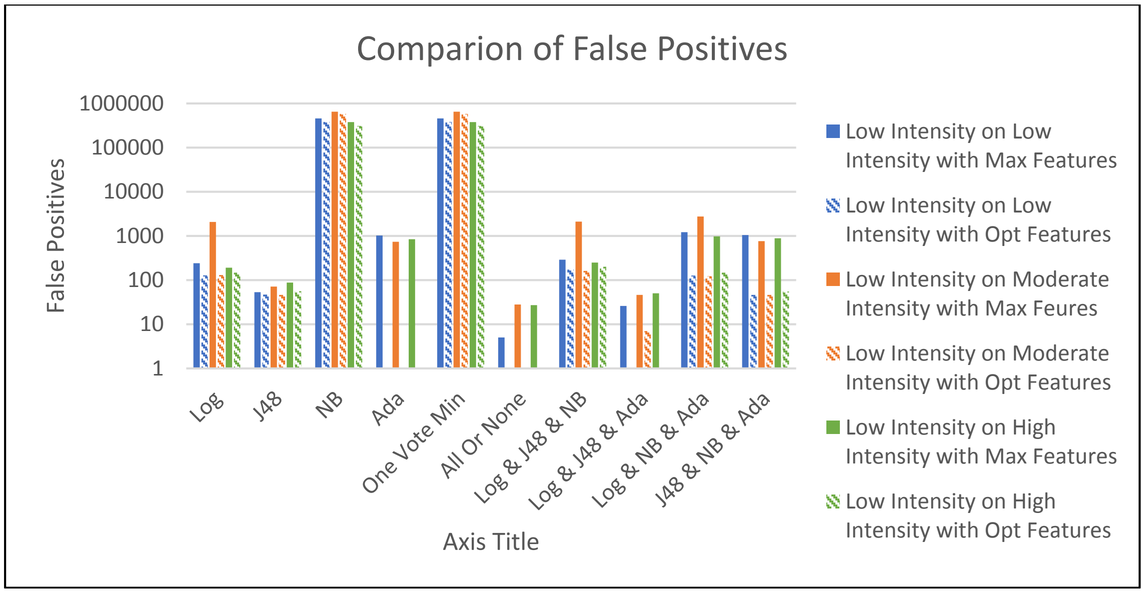


Figure 8. False Positive Comparison of Models Trained on Low Intensity Datasets

### 2. Logistic Regression

The Logistic Regression algorithm performed well, but never led any of the tests as the highest performing algorithm. It also never trailed as the poorest performing. Among the individual algorithms it scored the second best in all metrics except recall, where it was the second worst.

### 3. AdaBoost

The AdaBoost model performed well, but interestingly it did not out-perform the J48. As a boosting algorithm, intuitively it should be better than any of these other algorithms. One interesting anomaly of the AdaBoost algorithm was that the model trained on the low intensity dataset with optimized features never classifies any flow as botnet across any of the sets it is tested against. This is an example of an algorithm not having enough examples of a class to make a conclusion. Since only 0.5% of the training data was classified as botnet, the AdaBoost algorithm did not have enough to go on. Interestingly though, this only happened with the optimized feature set as seen in Table 10 and Table 11. Because using the maximum feature set increases the amount of data the algorithm has to work with, this was enough for it to make some conclusions of botnet traffic when trained under these conditions.

Table 10. Results for AdaBoost Algorithm (1 of 2)

| Testing Set | Low Intensity | | | | | | Moderate Intensity | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Training Set | Low Intensity | | Moderate Intensity | | High Intensity | | Low Intensity | | Moderate Intensity | | High Intensity | |
| Feature Set | Max | Opt | Max | Opt | Max | Opt | Max | Opt | Max | Opt | Max | Opt |
| ML Algorithm(s) | Ada | | | | | | | | | | | |
| Correctly Classified Instances (Qty) | 1668995 | 1662532 | 1668148 | 1657249 | 1654428 | 1655110 | 2413155 | 2359434 | 2413115 | 2367845 | 2403988 | 2395822 |
| Correctly Classified Instances (%) | 0.998856 | 0.994988 | 0.998349 | 0.991827 | 0.990138 | 0.990546 | 0.996914 | 0.974721 | 0.996898 | 0.978196 | 0.993127 | 0.989754 |
| Incorrectly Classified Instances (Qty) | 1911 | 8374 | 2758 | 13657 | 16478 | 15796 | 7469 | 61190 | 7509 | 52779 | 16636 | 24802 |
| Incorrectly Classified Instances (%) | 0.001144 | 0.005012 | 0.001651 | 0.008173 | 0.009862 | 0.009454 | 0.003086 | 0.025279 | 0.003102 | 0.021804 | 0.006873 | 0.010246 |
| Mean absolute error | 0.0011 | 0.005 | 0.0017 | 0.0082 | 0.0099 | 0.0095 | 0.0031 | 0.0253 | 0.0031 | 0.0218 | 0.0069 | 0.0102 |
| Root mean squared error | 0.0338 | 0.0708 | 0.0406 | 0.0904 | 0.0993 | 0.0972 | 0.0555 | 0.159 | 0.0557 | 0.1477 | 0.0829 | 0.1012 |
| Precision | 0.8801 | 0 | 0.768 | 0.2417 | 0.3369 | 0.3422 | 0.9867 | 0 | 0.9174 | 0.676 | 0.7871 | 0.7224 |
| Recall | 0.8935 | 0 | 0.9608 | 0.2952 | 0.9996 | 0.9613 | 0.8899 | 0 | 0.9641 | 0.2639 | 0.9981 | 0.9659 |
| F-Measure | 0.8868 | 0 | 0.8537 | 0.2658 | 0.504 | 0.5048 | 0.9358 | 0 | 0.9402 | 0.3796 | 0.8801 | 0.8266 |
| True Positives (TP) | 7482 | 0 | 8046 | 2472 | 8371 | 8050 | 54455 | 0 | 58995 | 16150 | 61076 | 59101 |
| True Negatives (TN) | 1661513 | 1662532 | 1660102 | 1654777 | 1646057 | 1647060 | 2358700 | 2359434 | 2354120 | 2351695 | 2342912 | 2336721 |
| False Positives (FP) | 1019 | 0 | 2430 | 7755 | 16475 | 15472 | 734 | 0 | 5314 | 7739 | 16522 | 22713 |
| False Negatives (FN) | 892 | 8374 | 328 | 5902 | 3 | 324 | 6735 | 61190 | 2195 | 45040 | 114 | 2089 |

Table 11. Results for AdaBoost Algorithm (2 of 2)

| Testing Set | High Intensity | | | | | | Ultra-High Intensity | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Training Set | Low Intensity | | Moderate Intensity | | High Intensity | | Low Intensity | | Moderate Intensity | | High Intensity | |
| Feature Set | Max | Opt | Max | Opt | Max | Opt | Max | Opt | Max | Opt | Max | Opt |
| ML Algorithm(s) | Ada | | | | | | | | | | | |
| Correctly Classified Instances (Qty) | 2045947 | 1359409 | 2096579 | 1582860 | 2112616 | 2101506 | 9300283 | 1469353 | 9880588 | 4181685 | 10191396 | 10062982 |
| Correctly Classified Instances (%) | 0.956459 | 0.63551 | 0.980129 | 0.739971 | 0.987626 | 0.982432 | 0.892962 | 0.141079 | 0.948679 | 0.401502 | 0.978521 | 0.966192 |
| Incorrectly Classified Instances (Qty) | 93138 | 779676 | 42506 | 556225 | 26469 | 37579 | 1114816 | 8945746 | 534511 | 6233414 | 223703 | 352117 |
| Incorrectly Classified Instances (%) | 0.043541 | 0.36449 | 0.019871 | 0.260029 | 0.012374 | 0.017568 | 0.107038 | 0.858921 | 0.051321 | 0.598498 | 0.021479 | 0.033808 |
| Mean absolute error | 0.0435 | 0.3645 | 0.0199 | 0.26 | 0.0124 | 0.0176 | 0.107 | 0.8589 | 0.0513 | 0.5985 | 0.0215 | 0.0338 |
| Root mean squared error | 0.2087 | 0.6037 | 0.141 | 0.5099 | 0.1112 | 0.1325 | 0.3272 | 0.9268 | 0.2265 | 0.7736 | 0.1466 | 0.1839 |
| Precision | 0.9988 | 0 | 0.9979 | 0.9724 | 0.9831 | 0.9878 | 0.9998 | 0 | 0.9995 | 0.9976 | 0.9965 | 0.9983 |
| Recall | 0.8816 | 0 | 0.9474 | 0.295 | 0.983 | 0.9637 | 0.8756 | 0 | 0.9407 | 0.3039 | 0.9784 | 0.9623 |
| F-Measure | 0.9366 | 0 | 0.972 | 0.4526 | 0.983 | 0.9756 | 0.9336 | 0 | 0.9692 | 0.4659 | 0.9874 | 0.98 |
| True Positives (TP) | 687381 | 0 | 738691 | 229982 | 766404 | 751377 | 7832541 | 0 | 8415494 | 2718924 | 8752373 | 8608534 |
| True Negatives (TN) | 1358566 | 1359409 | 1357888 | 1352878 | 1346212 | 1350129 | 1467742 | 1469353 | 1465094 | 1462761 | 1439023 | 1454448 |
| False Positives (FP) | 843 | 0 | 1521 | 6531 | 13197 | 9280 | 1611 | 0 | 4259 | 6592 | 30330 | 14905 |
| False Negatives (FN) | 92295 | 779676 | 40985 | 549694 | 13272 | 28299 | 1113205 | 8945746 | 530252 | 6226822 | 193373 | 337212 |

### 4. J48 Decision Tree

The J48 decision tree was the best performing algorithm among all ten combinations of voting methods in six of the seven metrics as seen in Figure 8. The only exception was recall, in which it scored the second best overall just behind the one vote minimum method.

For tests on the low intensity dataset, the J48 model trained on the low intensity dataset with maximum features performed the best for correctly classified instances, mean absolute error, root mean squared error, and F-measure. It also contributed to the best precision when used in majority voting with Logistic Regression and Naïve Bayes (with optimized features trained on low intensity). When testing on the moderate intensity dataset, the J48 model trained on the moderate intensity dataset with optimized features performed the best on all of the above metrics in addition to recall. The same holds true for testing on the high intensity dataset with the J48 model trained on the high intensity dataset with maximum features taking first place in these metrics.

Figure 9 shows a graphic visualization of the tree trained and tested on the moderate intensity dataset with maximum features. Color coding marks how many predictions are made at a particular leaf, with intervals increasing by an order of magnitude with each progressive shade. The errors in the algorithm are all confined to just three of the 30 leaves, annotated with slashed backgrounds instead of solid fills.

For comparison, Figure 10 shows the same J48 tree tested and trained on the same dataset but with the optimized feature set. This visual representation is valuable because it reveals aspects of the model not clearly seen otherwise. Since it can no longer use packets or bytes, it instead turns to the “A” flag (TCP Acknowledgement) as the root of the tree.

With both models, the source and destination port numbers were highly valuable as decision points, used throughout both trees. Also interesting was that the optimized feature set results in a larger tree, growing from 39 to 43 nodes, 20 to 22 leaves, and a depth of 10 to 11 levels.

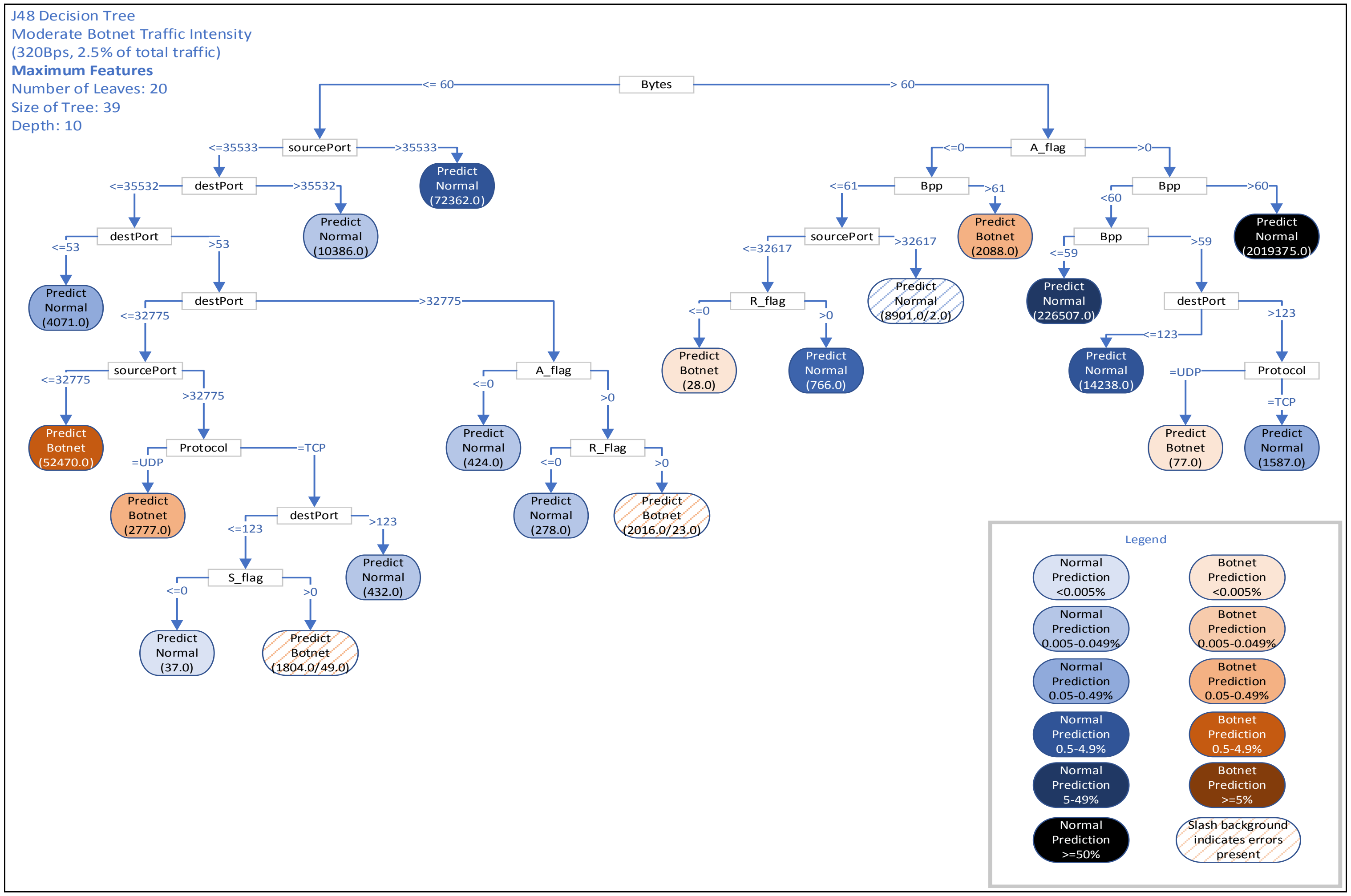


Figure 9. J48 Decision Tree with Maximum Features Trained and Tested on Low Intensity Dataset

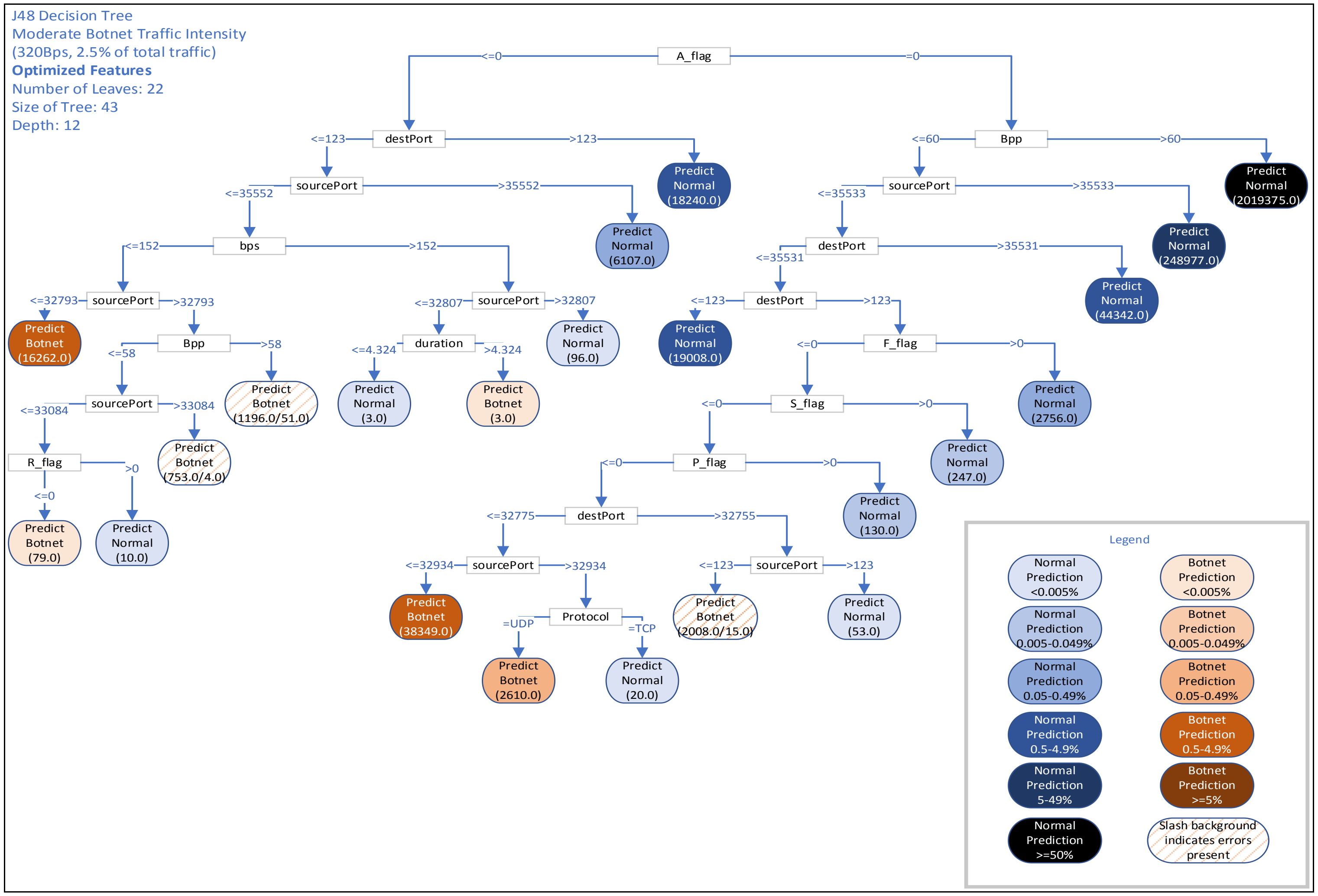


Figure 10. J48 Decision Tree with Optimized Features Trained and Tested on Low Intensity Dataset

## F. VOTING METHODS

While the intent of using voting methods is to improve performance, our results showed that this is not always the case. In addition to the metrics described below, there is the added overhead of time and computing resources necessary to use these methods. To take a majority vote between three algorithms, you must build models for all three, run the test data through all three models, and then conduct the voting from each algorithm's predictions. This is a non-trivial addition to the process, especially when you consider scaling up these methods to handle higher bandwidth networks. This is all to say that even if a marginal improvement can be found through a voting method, its improvement must be weighed against the increased cost of implementation.

### 1. All or None

The all or none voting method performed better than the Naïve Bayes and AdaBoost models on their own in six of the seven metrics. The only exception was recall, in which it performed the worst of all ten combinations of algorithms tested. This makes sense because recall is the measurement of what percent of positive results were found. By requiring consensus of all voting models to predict a positive instance, this method actually suffers by being bound to any false negative in any algorithm. Unfortunately for this voting method, recall is a particularly important metric in this application as we are trying to identify all the malicious flows we can.

### 2. One Vote Minimum

The one vote minimum method suffered greatly from the poor performance of the Naïve Bayes algorithm. Since Naïve Bayes had a substantially larger number of false positives, this brought down the one vote minimum method. It performed the worst among all ten algorithm combinations in six of the seven metrics. The exception was recall, in which it actually achieved the highest performance of all ten combinations. This lines up with the fundamental definition of the one vote minimum scheme which ensures the greatest number of positives are identified by counting any individual algorithm's positive verdict as enough justification. From our particular application of defending a network, this high recall alone could make it an excellent contender for real world implementation.

However, the increased cost must be considered as it requires the training and testing of all four algorithms followed by the voting step itself. Still, if recall is truly the most important metric to a network administrator, that may be justification enough to warrant use of this method.

#### 3. Majority

In order to evaluate a manageable set of results, we used the four unique combinations of three algorithms each to conduct simple majority voting. All four of these resulted in better performance across all seven metrics than the Naïve Bayes and AdaBoost models on their own with the sole exception of Naïve Bayes' recall.

Among these four combinations, Figure 8 shows how majority voting between Logistic Regression, J48, and Naïve Bayes had the second-best performance out of all ten algorithm combinations in the categories of percent correctly classified, percent incorrectly classified, mean absolute error, and root mean squared error. It also ranked three out of ten in recall and F-measure. These results are somewhat surprising when accounting for the poor performance of Naïve Bayes on its own. Intuitively, one would assume that because Naïve Bayes was the worst single algorithm, that the best majority voting method would be the one to exclude it. Our results showed that this intuition was incorrect. In fact, while the majority voting among Logistic Regression, J48, and Naïve Bayes did not rank in the lowest three of any metric, it did score the second-place position for both precision and F-measure. Also of note was the majority voting among J48, Naïve Bayes, and AdaBoost ranked third of ten in the metrics of correctly classified, incorrectly classified, mean absolute error, and root mean square error.

## G. DISPARITY OF TRAINING AND TESTING DATASETS

It is a common-sense assumption to believe that the model trained on a particular intensity dataset should perform best when tested on that same intensity dataset. This holds true in the results we collected. In almost every case, the best performing metric was seen in the scenarios with models trained and tested on the same intensity level dataset. For testing on the low intensity data, all but three of the thirteen metrics had their peak with the models trained on the same dataset. It was only recall, true positives, and false negatives that were better when using the models trained on the other datasets. For testing on moderate intensity datasets, there were also only three of the thirteen metrics that fared better on models with training on other datasets. This time they were precision, true negatives, and false positives. The same held true for the testing of high intensity datasets, with the same three exceptions. It should also be noted that the best F-measure, perhaps the most comprehensive single metric, always occurred in the models trained with the same intensity dataset. Even more revealing is that it was always the J48 algorithm with maximum feature set that achieved this coveted spot.

It is another fairly easy assumption to make that the models trained on datasets with more examples of botnet traffic should have a better understanding of the classification and therefore perform better. This appears to hold true for our results as well.

## H. NETWORK ADMINISTRATOR REPORT GENERATION

To maximize the usefulness of our results, we implemented a report generation process at the end of Phase 2. Since the outputs of Weka’s evaluation are a series of predictions on individual flows, it was important to translate this into usable information for a network administrator. The resulting report shows the administrator the IP addresses of the most suspected flows. For each IP address, it reports the number of flows classified as normal along with the average percentage confidence in those normal predictions. It then does the same for the number of flows predicted as botnet and the average percentage confidence of those predictions. The list is sorted by highest number of botnet predictions and is separated into source and destination IP addresses. This split is to help the administrator understand whether the suspected malicious flows are going to or coming from his network. Since the software has no sense of the network topology, this is where the administrator’s knowledge of their network can be leveraged. Armed with this report, they can prioritize their efforts and allocate resources accordingly. Figure 11 is one such report generated for evaluating the low intensity dataset using the J48 model trained on the moderate intensity dataset with maximum features. Of note, the infected machine in our emulation was 10.0.0.65 and the target machine was 10.10.50.28. This report shows that 5,598 flows with 10.0.0.65 as the source IP were classified by the model as botnet traffic and the average confidence in those predictions is 1.0 (rounding effects presents). To the operator, this is much more valuable information than the Weka output showing millions of instances with a prediction for each.

```
Report for: J48 Model Trained with Maximum Features on Moderate
Intensity Dataset Tested on Low Intensity Dataset

***SRCIPs***
10.0.0.65          ['Normal', 58315, 1.0]     ['Botnet', 5598, 1.0]
10.10.50.28        ['Normal', 1, 1.0]         ['Botnet', 2781, 1.0]
10.0.0.90          ['Normal', 119601, 1.0]    ['Botnet', 91, 0.97]
10.0.0.22          ['Normal', 18530, 1.0]     ['Botnet', 57, 0.97]
10.0.0.113         ['Normal', 391688, 1.0]    ['Botnet', 35, 1.0]
10.0.0.60          ['Normal', 86413, 1.0]     ['Botnet', 17, 0.97]
10.0.0.25          ['Normal', 75736, 1.0]     ['Botnet', 10, 0.98]
10.0.0.82          ['Normal', 52701, 1.0]     ['Botnet', 10, 0.97]
10.0.0.26          ['Normal', 73703, 1.0]     ['Botnet', 10, 0.97]
10.0.0.81          ['Normal', 15021, 1.0]     ['Botnet', 8, 0.97]
10.0.0.21          ['Normal', 62794, 1.0]     ['Botnet', 8, 0.97]
10.0.0.1           ['Normal', 26870, 1.0]     ['Botnet', 2, 0.97]
10.0.0.11          ['Normal', 16075, 1.0]     ['Botnet', 2, 0.97]
195.20.250.172     ['Normal', 630, 1.0]       ['Botnet', 2, 0.99]
74.208.191.197     ['Normal', 86, 1.0]        ['Botnet', 2, 0.99]
172.217.3.164      ['Normal', 6346, 1.0]      ['Botnet', 1, 1.0]
172.217.3.196      ['Normal', 5273, 1.0]      ['Botnet', 1, 1.0]
208.91.197.27      ['Normal', 4, 1.0]         ['Botnet', 1, 0.99]
23.111.9.30        ['Normal', 171, 1.0]       ['Botnet', 0, 0]
72.167.18.239      ['Normal', 9353, 1.0]      ['Botnet', 0, 0]

***DSTIPs***
10.10.50.28        ['Normal', 0, 0]           ['Botnet', 5592, 1.0]
10.0.0.65          ['Normal', 57126, 1.0]     ['Botnet', 2781, 1.0]
198.189.255.153    ['Normal', 1490, 1.0]      ['Botnet', 145, 0.97]
198.189.255.140    ['Normal', 2613, 1.0]      ['Botnet', 46, 0.97]
10.0.0.60          ['Normal', 85016, 1.0]     ['Botnet', 10, 1.0]
198.189.255.162    ['Normal', 514, 1.0]       ['Botnet', 10, 0.97]
10.0.0.90          ['Normal', 118246, 1.0]    ['Botnet', 9, 1.0]
10.0.0.25          ['Normal', 74113, 1.0]     ['Botnet', 8, 1.0]
10.0.0.26          ['Normal', 72163, 1.0]     ['Botnet', 7, 1.0]
173.241.250.220    ['Normal', 6089, 1.0]      ['Botnet', 6, 0.98]
172.217.0.46       ['Normal', 11552, 1.0]     ['Botnet', 6, 0.97]
10.0.0.82          ['Normal', 51666, 1.0]     ['Botnet', 4, 0.99]
35.171.222.21      ['Normal', 20, 1.0]        ['Botnet', 4, 0.97]
178.255.83.1       ['Normal', 4125, 1.0]      ['Botnet', 3, 0.97]
10.0.0.11          ['Normal', 15663, 1.0]     ['Botnet', 2, 0.99]
10.0.0.21          ['Normal', 61752, 1.0]     ['Botnet', 2, 0.99]
62.201.164.117     ['Normal', 590, 1.0]       ['Botnet', 1, 0.97]
10.0.0.113         ['Normal', 392133, 1.0]    ['Botnet', 0, 0]
10.0.0.1           ['Normal', 26417, 1.0]     ['Botnet', 0, 0]
23.111.9.30        ['Normal', 171, 1.0]       ['Botnet', 0, 0]
```

Figure 11. Results Report for J48 Model Trained on Moderate Intensity Dataset and Applied to Low Intensity Dataset

THIS PAGE INTENTIONALLY LEFT BLANK

# V. CONCLUSION AND FUTURE WORK

## A. CONCLUSION

Our research shows that machine-learning can be a valuable tool leveraged to enhance network security in small USMC tactical networks. In our emulated network, the J48 decision tree in particular showed impressive results across all pertinent metrics. Furthermore, we showed that different algorithms and voting techniques have varying levels of performance, and therefore the most efficient methods can be found and implemented as layers of security for different applications. Furthermore, our results indicate that machine-learning methods can adapt to a network's unique traffic and identify botnet traffic based on discriminating features without requiring prior traffic signatures.

The experimental design decisions support the conclusion that machine-learning can be applied to currently fielded DoD hardware of tactical networks. More so, the software utilized is all open-source and well documented in use by many reputable universities. We consider all the software utilized in this work to be mature and stable. Because of these conditions, the implementation of machine-learning as a network security tool would indeed be a low-cost solution.

It is important to note that our research in no way proved or suggested that machine-learning was a replacement to any network security technique currently employed by the DoD. However, we did show that these machine-learning techniques can identify malicious traffic in a completely different way than current firewalls operate, thus providing a more robust solution if paired together.

Of note, our research showed this approach is not just an academic endeavor, but that it can provide network administrators with reports on their network that are clear, easy to understand, and most importantly actionable. We do acknowledge that this research did not provide a single software solution, but rather a methodology for implementing multiple components together. Additionally, we acknowledge while machine-learning methods benefit from not requiring prior signatures, they do require training on labeled datasets. This collection and labeling of training sets is not trivial and does present a limitation for application of this methodology at present. The methods researched here would have to be developed further into a more user-friendly final product to allow operators the ability to use such tools without needing extensive understanding of the inner mechanisms. We do not propose fielding any solution that would require extensive additional training and education of network administrators, and so additional research in this area is needed.

## B. FUTURE WORK

While this research set up a unique test network and covered experiments with many variables, there are some key areas that could provide interesting and useful results if examined further.

### 1. Analyze Additional Botnet Traffic Types

This research was limited to a single use case for malicious activity by a botnet: DDoS. Today's botnets are used for a whole host of malicious purposes, so further work with accurate simulators would be useful. The most common and therefore likely most fruitful would be bots that send spam, exfiltrate data, or commit click fraud. Additionally, it would be enlightening to see this moved beyond simulators and actually tested on systems infected with actual malware. This would have to be carefully controlled and understood to ensure the botnet malware does not replicate and infect other hosts on the network without specific commands from the researcher.

### 2. Increase Scale

The decision to use an emulated network with actual hardware on a live network in real time was a major feature of this research, but it does present issues for scaling. If an efficient method for increasing the scale of this emulated network could be implemented, it would be valuable to see if similar results are achieved. Because of the computational limitations discovered in this work with the ultra-high botnet intensity dataset, any increase in scale would increase the demand for computational resources. These resources, however, are limited due to the nature of seeking solutions for tactical networks with their current hardware. To balance the competing requirement of processing power and scale, it may be possible to develop a scheme to capture NetFlow data from smaller time periods so that the resulting number of instances is able to be processed by Weka. This would need to be explored further for its feasibility and to investigate whether the ML models achieve a comparable level of success.

### 3. Assess Usability with USMC Network Administrators

This work included a key step in the end of Phase 2 with the report generation. This is in keeping with the desired purpose of exploring solutions that could be implemented with current technology. A critical follow on opportunity would be to package the tools developed here along with user training material. The resultant package could be provided to a few actual USMC network administrators and or information assurance Marines. The Marines could attempt to implement the tools and create models on a test network. the researcher could then inject botnet traffic in a controlled manner. After applying the trained algorithm to the live test network, the Marines could attempt to identify any suspected botnet machines on the network from the report. The researcher could measure how effective the training material is, how difficult the tools are to learn and implement, and any other relevant feedback. This would also seek to determine if the processing capacity available to these administrators in their tactical networks is sufficient for generating the ML models and using them to classify their traffic.

### 4. Compare Results to Firewall Protected Network

This research sought to discover the effectiveness of identifying malicious traffic through non-traditional applications of machine-learning. It would be beneficial to compare these results with the current traditional methods of perimeter firewalls. This would seek to discover whether the inclusion of ML based detection methods added unique or redundant security measures to the DoD’s current network security defense in depth.

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES

[1] Symantec, "Internet security threat report (Volume 23)," 2018. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf

[2] A. L. Buczak and E. Guven, "A survey of data mining and machine-learning methods for cyber security intrusion detection," IEEE Communications Surveys & Tutorials 18.2: 1153-1176., 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7307098/

[3] W. T. Strayer, D. Lapsely, R. Walsh and C. Livadas, "Botnet detection based on network behavior," Springer, Boston, MA, 2008. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-0-387-68768-1_1.pdf

[4] R. Thormeyer, "Hacker arrested for breaching DoD systems with ‘botnets’," Government Computer News, 4 November 2005. [Online]. Available: https://gcn.com/articles/2005/11/04/hacker-arrested-for-breaching-dod-systems-with-botnets.aspx.

[5] The Secretary of Commerce and The Secretary of Homeland Security, "A report to the president on enhancing the resilience of the internet and communications ecosystem against botnets and other automated, distributed threats," 2018. [Online]. Available: https://csrc.nist.gov/publications/detail/white-paper/2018/05/30/enhancing-resilience-against-botnets--report-to-the-president/final

[6] President of the United States, "Executive Order 13800: Strengthening the cybersecurity of federal networks and critical infrastructure," 11 5 2017. [Online]. Available: https://www.federalregister.gov/documents/2017/05/16/2017-10004/strengthening-the-cybersecurity-of-federal-networks-and-critical-infrastructure.

[7] Defense Advanced Research Projects Agency, "Harnessing autonomy for countering cyberadversary systems (HACCS)," 3 8 2017. [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&id=e37dc8983aa4347361744a3cfbb43ec5&tab=core&_cview=0.

[8] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, M. Hiroshi, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand and D. Steinberg, "Top 10 algorithms in data mining," Springer-Verlag London Limited, London, UK, 2007. [Online]. Available: http://www.cs.uvm.edu/~icdm/algorithms/10Algorithms-08.pdf

[9] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," IEEE Communications Surveys & Tutorials 10.4: 56-76., 2008. [Online]. Available: https://ieeexplore.ieee.org/document/4738466/

[10] I. H. Witten, E. Frank and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd Ed, San Francisco: Morgan Kaufmann, 2011.

[11] Machine-Learning Group at the University of Waikato, "https://www.cs.waikato.ac.nz/ml/weka/," 2018. [Online].

[12] S. Garcia, M. Grill, J. Stiborek and A. Zunino, "An empirical comparison of botnet detection methods," Czech Technical University, Prague, Czech Republic, 2014. [Online]. Available: https://dl.acm.org/citation.cfm?id=2665897

[13] F. Flock, "K-folds cross-validation visualization," 20 September 2016. [Online]. Available: https://commons.wikimedia.org/wiki/File:K-fold_cross_validation_EN.jpg.

[14] Markus-Go, "BoNeSi - The DDoS Botnet Simulator," 25 1 2016. [Online]. Available: https://github.com/Markus-Go/bonesi.

[15] A. Tarem, B. Oreshkin and M. Coates, "Machine-learning approaches to network anomaly detection," Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques. USENIX Association, 2007. [Online]. Available: https://dl.acm.org/citation.cfm?id=1361449

[16] G. Kakavelakis, "Auto-learning of SMTP TCP transport-layer features for spam and abusive message detection," Naval Postgraduate School, Monterey, CA, 2011. [Online]. Available: https://calhoun.nps.edu/handle/10945/36368

[17] B. Li, J. Springer, G. Bebis and M. H. Gunes, "A survey of network flow applications," Journal of Network and Computer Applications 36.2: 567-581., 2013. [Online]. Available: https://dl.acm.org/citation.cfm?id=2444244

[18] J. Zhang, J. Tang, X. Zhang, W. Ouyang and D. Wang, "A survey of network traffic generation," Beijing, China, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7446954/

[19] R. W. Becker, "A test bed for detection of botnet infections in low data rate tactical networks," Naval Postgraduate School, Monterey, CA, 2009. [Online]. Available: https://calhoun.nps.edu/handle/10945/4650

[20] K. Beneduce, "Attributes and machine-learning for fragment identification and malware analysis," Naval Postgraduate School, Monterey, CA, 2014. [Online]. Available: https://calhoun.nps.edu/handle/10945/48126

[21] M. R. Thakur, D. R. Khilnani, K. Gupta, S. Jain and V. Agarwal, "Detection and prevention of botnets and malware in an enterprise network," International Journal of Wireless and Mobile Computing 5.2: 144-153, 2012. [Online]. Available: https://arxiv.org/abs/1312.1629

[22] J. Wang and I. C. Paschalidis, "Botnet detection based on anomaly and community detection," IEEE Transactions on Control of Network Systems 4.2: 392-404., 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7422020/

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