View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AFTI Scholar (Air Force Institute of Technology)

Air Force Institute of Technology
AFIT Scholar

Theses and Dissertations

Student Graduate Works

9-15-2011

A Novel Malware Target Recognition Architecture for Enhanced Cyberspace Situation Awareness

Thomas E. Dube

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Digital Communications and Networking Commons, and the Information Security Commons

Recommended Citation

Dube, Thomas E., "A Novel Malware Target Recognition Architecture for Enhanced Cyberspace Situation Awareness" (2011). *Theses and Dissertations*. 1379.
https://scholar.afit.edu/etd/1379

This Dissertation is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.

A NOVEL MALWARE TARGET RECOGNITION ARCHITECTURE FOR ENHANCED CYBERSPACE SITUATION AWARENESS

DISSERTATION

Thomas E. Dube, Major, USAF

AFIT/DCE/ENG/11-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT/DCE/ENG/11-07

A NOVEL MALWARE TARGET RECOGNITION ARCHITECTURE FOR ENHANCED CYBERSPACE SITUATION AWARENESS

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Thomas E. Dube, B.C.E., M.S.
Major, USAF

September 2011

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/DCE/ENG/11-07

A NOVEL MALWARE TARGET RECOGNITION ARCHITECTURE FOR ENHANCED CYBERSPACE SITUATION AWARENESS

Thomas E. Dube, B.C.E., M.S.
Major, USAF

Approved:

[illegible] 10 Aug 11

Dr. Richard A. Raines
Dissertation Advisor
Date

[illegible] 10 AUG 11

Dr. Kenneth W. Bauer
Committee Member
Date

[illegible] 10 AUG 11

Dr. Michael R. Grimaila
Committee Member
Date

[illegible] 10 Aug 11

Dr. Steven K. Rogers
Committee Member
Date

Accepted:

MU Thomas 9 Sep 11

M. U. Thomas
Dean, Graduate School of Engineering and Management
Date

AFIT/DCE/ENG/11-07

Abstract

The rapid transition of critical business processes to computer networks potentially exposes organizations to digital theft or corruption by advanced competitors. One tool to steal company secrets or manipulate information is malware. Malware circumvents legitimate authentication mechanisms and is an epidemic problem for organizations of all types, including governments, militaries, sectors of critical infrastructure and businesses.

This research proposes, designs, implements and evaluates a novel Malware Target Recognition (MaTR) architecture for malware detection and identification of propagation methods and payloads to enhance situation awareness in tactical scenarios using non-instruction-based, static heuristic features with standard machine learning algorithms. Recent published research in static heuristics focuses on detection using n-grams as features, which are computationally determined, short n-byte sequences that are resource intensive to compute and directly unintelligible to human operators. MaTR achieves a 99.92% detection accuracy on known malware with false positive and false negative rates of 8.73e-4 and 8.03e-4 respectively.

In comparison, MaTR outperforms leading n-gram methods with a statistically significant 1% improvement in detection accuracy against known malware and 85% and 94% reductions in false positive and false negative rates respectively. Against a set of publicly unknown malware, MaTR detection accuracy is 98.56%, a 3.8% engineering advantage over n-gram methods and a 65% performance improvement over the combined effectiveness of three commercial antivirus products (both statistically significant). MaTR identification of propagation methods and payloads are greater than 86% and 83% respectively, which is comparable to existing research, but relies on simpler features to collect allowing for efficient retraining and redeployment. Collectively, MaTR classifiers provide a significant improvement over existing technologies and enable operators to achieve higher levels of situation awareness in cyberspace.

Acknowledgments

And I gave my heart to seek and search out by wisdom concerning all things that are done under heaven: this sore travail hath God given to the sons of man to be exercised therewith. (Eccl. 1:13)

I would like to thank my advisor for his advice and mentorship as I pursued my graduate research. He gave me the liberty to pursue my interests in an "unlikely" area for successful research. I also thank my committee members, whose critical feedback provided additional perspective and significantly improved my work and its presentation.

My heartfelt thanks go to my family for their sacrifices as I embarked on this long and challenging journey. Their prayers and support were critical to my success at many points along the way. I can never repay them for how much their undying love and unwavering support meant to me during this effort.

I am eternally indebted to my Heavenly Father for not only sending His Son, Jesus Christ, to die on the cross as the sacrifice for my sins that I shall be free from the judgment, but for His hand of guidance after I trusted in Him as my Savior. He completely changed my life and placed me on the path that I still walk today—and this academic accomplishment is just another step along that same path that He started me on over 18 years ago. With great anticipation, I long to see where His path leads me.

Thomas E. Dube

Table of Contents

Page

List of Figures

Figure Page

List of Tables

Table Page

Table Page

List of Abbreviations

Abbreviation		Page
AUC	area under curve	62
CRISP-DM	CRoss-Industry Standard Process for Data Mining	71
ANOVA	analysis of variance	85
TPR	true positive rate	106

A NOVEL MALWARE TARGET RECOGNITION ARCHITECTURE
FOR ENHANCED CYBERSPACE SITUATION AWARENESS

I. Introduction

The widespread adoption of networked Information and Communication Technologies (ICT) by all facets of society has made massive amounts of valuable information vulnerable to digital theft. As organizations and individuals embed ICT into their core operational processes, many have unwittingly exposed themselves to exploitation. The result is an extremely appealing target for competitors and a new wave of cyberspace criminals lured by easy profit and unlikely prosecution. More information is available today for surreptitious exploitation than ever before, while organizations continue to struggle with standard computer network defense (CND) practices and are only beginning to realize true attacker perceptions of their information's value.

1.1 Motivation

Malware, a portmanteau of "malicious software," is the cyberspace weapon system of choice enabling attackers to conduct a wide gamut of offensive information operations as evidenced by the now infamous Stuxnet worm [43]. The Stuxnet worm payload causes a loss of data integrity for supervisory control and data acquisition (SCADA) systems [43], which are systems that run industrial control systems, such as power grids. One of the most dangerous operations is data exfiltration, where the attacker increases their competitive edge by harvesting sensitive information from unsuspecting victims. Imagine the value and impact of obtaining blueprints for the most advanced jet fighter [35] at no substantial cost or obtaining millions of sensitive customer records [29].

Malware detection has been an active computer security research area for decades. Advances in this area have not produced a "silver bullet" solution to this problem, because it is ultimately a *human* enterprise. Solutions should enhance human effectiveness to defend the network, not replace them. A relatively small set of malware can hide amongst a million unique executables on large networks making it difficult for humans to find without a form of automated assistance.

With attacker motivation at an all-time high, customized malware attacks are becoming more common and allow adversaries to sidestep the traditional front-line defense, signature-based antivirus software [14, 15]. Antivirus software often fails to detect targeted threat tools that use similar methods to those described by [14], namely dead-code insertion, code transposition, register reassignment, and instruction substitution.

Cyberspace adversaries are adaptable foes, and methods to detect them must also adapt or risk becoming obsolete. This observation has produced unique research momentum for new detection technologies that do not require a continual stream of updated antivirus signatures [24, 87]. Generic detection technologies make extensive use of classic pattern recognition and machine learning techniques. If hackers can victimize governments and high-profile corporations by avoiding antivirus software, the risk to lesser-financed organizations is likely higher than perceived. Visibility into network activity is limited, because of the immense volume of data and the difficulties associated with effective data reduction.

1.1.1 Hypothetical Scenario.

In order to understand the current state of one's own network, one must first have sufficient situation awareness (SA). Endsley [32] defines SA as "the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future." With adequate SA, organizations can perceive previously unknown threats, comprehend threat capability and ultimately project future threat activity. Taken further, sufficient SA enables organizations to short circuit the impact of continuing threat activity.

Figure 1 shows a partial network diagram for a fictitious company with three major business divisions. In this example, malware exists on the circled systems. With traditional defenses, even this simple detection information is most likely unavailable to organizations—especially if their information piques the interest of competitive large corporations or nation-states. Published research finds that simple obfuscations are enough to thwart commercial antivirus signatures [14, 15].

A generic malware detection capability does not rely on a signature database for byte sequences [5, 87, 91] to make specific identifications or instructions [87, 91] to make static heuristic matches. A generic malware detection capability can illuminate a portion of the unknowns allowing an organization to detect threats. They may not know the threat capability, but they know the threat is present. The generic detection information corresponds to the first level of the Endsley SA model [33], perception. The information afforded by generic detection capabilities alone, however, is not necessarily actionable. In the situation from Figure 1, leadership cannot deduce threat intentions or next moves with detection information alone. Therefore, the decision making process can only consider vague information regarding possible threat intent and likely next targets.

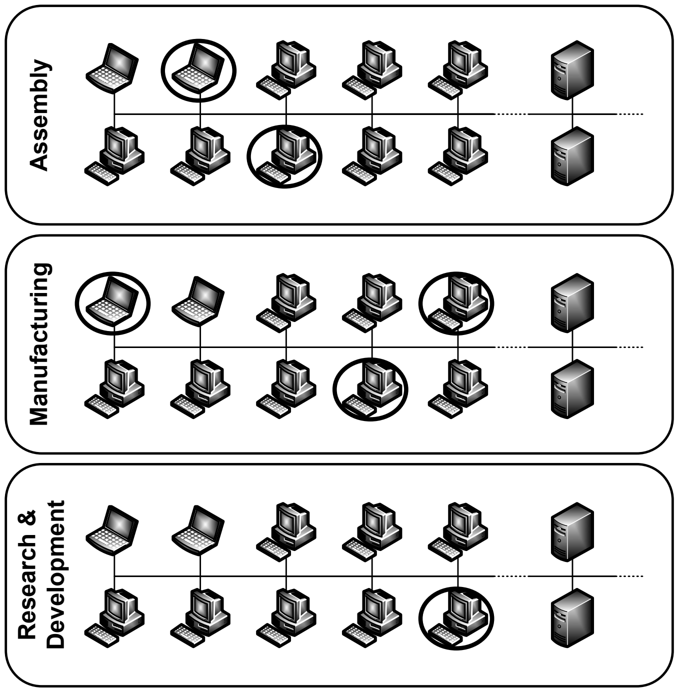

Figure 1. Situation awareness indicators afforded by malware detection.

1.1.2 Advanced Persistent Threat Motivation.

Christodorescu and Jha [14] summarize the conflict between malware authors and researchers as an "obfuscation-deobfuscation" game, where advances from either side prompt retaliatory responses to at least maintain the status quo. Although an excellent observation about the tactics employed between two major players, it falls short of casting the strategic level problem to the players on a different plane, specifically potential victims. An operational extension of their observation is the game of "cat-and-mouse" between organizations and advanced competitors that employ malware to maintain unauthorized access to sensitive information.

In this high-tech cat-and-mouse game, threats (the mice) have a major asymmetric advantage over the victim cats, because the cats are essentially blind, deaf, or halt with respect to minor changes in mouse tactics. Normally, commercial antivirus products (the mousetraps?) are the cat's most reliable indicator of mouse presence.

If these mice discover methods to beat the mousetrap [14, 15], though, they can steal the cheese. What is the "cheese" that the mice are after? They seek access to sensitive information, such as intellectual property, operations plans, personnel data, schedules, etc. Armed with this information, the mice ultimately hope to erode the cat's competitive advantage.

Christodorescu and Jha [14, 15] highlight difficulties commercial products have with handling simple obfuscation techniques, such as `nop` (a "no operation" assembly instruction) insertion and inserting unconditional branches (opaque predicates as described in [19]). In their test, they randomly apply these transformations to known malware samples that three commercial antivirus products initially detected. After transformation, they find that all three antivirus products failed to recognize ten unique mutations of the malware samples.

Competitive threats can easily employ similar techniques to make unique malware samples that victim defensive tools cannot detect. An advanced persistent threat (APT) is a nation-state- or large corporation-sponsored competitive threat that is capable and determined to accomplish its goals [8]. While malware is not the only method of gaining information at the APT's disposal, it can satisfy their operational needs for victim network access, data exfiltration and data corruption. According to Bejtlich [8], achieving cyberspace SA allows organizations to potentially discover and thwart APT operations. Major asymmetric advantages of the competitive threat include unauthorized access to competitor sensitive data, low likelihood of discovery and prosecution, and low tool development cost. Ethics aside, the return on investment of this employment strategy is extremely high. The strategic level shortcomings of the status quo are the motivations for this current research.

1.2 Research Goals

The major goal of this research is to significantly enhance cyberspace SA capabilities regarding malware threat activity by developing targeted machine learning classifiers that provide the operator with *relevant* threat information. Relevant threat information directly enhances the operator's ability to achieve higher levels of SA by identifying threat attributes, such as malware propagation methods and payloads. In addition to detection, the classifiers in the Malware Target Recognition (MaTR) architecture identify the risk of threat spread across the network and specific threat capabilities. As a measure of success, each targeted classifier should demonstrate superior performance to current research and commercial products that provide similar information.

These goals coincided with the vision of the Chief Scientist of the United States Air Force (USAF) when he previously introduced the concept of enterprise Focused Long Term Challenges (FLTCs) and identified where cyberspace research fits into the strategic vision for technology [20]. He indicated that the USAF cyberspace mission is burgeoning and becoming recognized as a vital element with far reaching effects on operations.

In turn, the Air Force Research Laboratory (AFRL) Executive Director previously attributed specific focus areas to the Air Force enterprise FLTCs [78]. The eight identified FLTCs (shown in Table 1) also relate to the specific focus areas in Table 2. This cyberspace research effort targets the CND and SA focus areas, which directly impact the following FLTCs: #1, #5, #7 and #8.

Arguably, the greatest impact of this research is in assurance of operational capability in high-threat environments (FLTC #5). This particular capability is important not only in wartime, but equally so in peacetime. The USAF has a great wealth of both operations and research information stored on government, industry

Table 1. **AFRL FLTCs for cyberspace.**

FLTC	**Name**
FLTC #1	Anticipatory Command, Control & Intelligence
FLTC #2	Unprecedented Proactive Intelligence, Surveillance & Reconnaissance
FLTC #3	Dominant Difficult Surface Target Engagement & Defeat
FLTC #4	Persistent & Responsive Precision Engagement
FLTC #5	Assured Operations in High Threat Environments
FLTC #6	Dominant Offensive Cyber Engagement
FLTC #7	On-demand Force Projection, Anywhere
FLTC #8	Affordable Mission Generation & Sustainment

Table 2. **AFRL FLTCs focus areas for cyberspace.**

Focus Area
Computer Network Defense
Command and Control
Situational Awareness
Electronic Warfare
Cyber Network Attack
Cyber Exploitation

and academic networks—much of which is beyond the direct control and protection of the USAF enterprise. A compromise of key weapon system information not only jeopardizes future capabilities, but potentially wastes an exorbitant amount of the nation's capital. The impact of this research may not only preserve current enterprise operations and national fiscal resources, but also future weapon systems and operations.

Researchers and information assurance experts widely criticize major current CND technologies, specifically antivirus and intrusion detection systems (IDS), as inadequate to meet security requirements. Such solutions are prone to fail at detecting novel or adapted threats [14, 24, 65]. Their coarse outputs have questionable effectiveness as indicators for cyberspace SA, because the provided information is not actionable until confirmed or combined with other sources. For instance, an antivirus product may alert an operator of the presence of malware *ABC*, but the operator must often research *ABC*'s functionality for damage assessment. Only when combined with this description does the discovery of malware *ABC* become actionable. The MaTR architecture directly provides additional, relevant threat information to the operator without casting additional manual processes on the user.

This research provides a potentially disruptive technology in malware detection with additional threat information, which can provide the USAF, the Department of Defense and the United States government with an asymmetric advantage in CND. This technology has the potential to identify previously unknown, custom malware attacks that can cripple data networks and the operations which rely on their continued presence. Coupling this detection information with additional, targeted threat information, such as malware propagation methods and payloads, enhances the ability of experienced operators to achieve SA in cyberspace.

1.3 Research Contribution

Static analysis of malware does not require central processing unit (CPU) emulation [87, 91]. "Heuristic" has a Greek root and means "to discover", but in malware research it refers to algorithms that provide suboptimal solutions to complex problems [87]. Static heuristics are heuristic solutions obtained through static analysis. Current static heuristic malware research focuses on using the following feature sources for detection:

1. strings [17, 77],
2. structure [72, 77, 91],
3. anomalies [72, 77, 91, 96],
4. instructions [9, 14, 15, 16],

5. application programmer interface (API) calls [9, 77, 85, 91],

6. control flow graphs [9, 16], and

7. n-grams [1, 5, 46, 47, 77, 92, 94].

A malware author can easily manipulate strings to a avoid a string-based detection method [77]. Items 4 through 6 require a correct disassembly of program instructions to make determinations, which is a difficult (if not impossible) task [65]. The same way attackers can manipulate strings to avoid detection [77], they can manipulate n-grams as well. On the other hand, manipulation of structure and anomalies may require avoiding common malware defenses, such as packing. This research examines the effectiveness of a proposed architecture using only anomaly and structural features to detect malware. Other static heuristic research [72, 77, 96], which uses similar features to MaTR, does not achieve the same level of performance. The proposed MaTR architecture achieves a statistically significant improvement over the remaining primary non-instruction-based static heuristic, n-grams.

Additionally, this research expands the application problem of non-instruction-based, static heuristic features to the identification of malware propagation methods and payloads. These observations are key components to framing threat context and producing operationally *actionable* information. Few researchers examine similar applications with static heuristics [47], which is critical threat information to enhance cyberspace SA. The MaTR prototype has higher mean performance than comparable research for malware propagation methods and payloads.

Finally, a mapping of experimental results to Endsley's SA model [33] for a typical scenario demonstrates how this research leads to a heightened awareness of threat activity. This SA, as explained later in detail, ultimately gives operators and leadership clearer information on which to base strategic and tactical decisions. No other known malware research includes an extensive operational mapping to SA models.

1.4 Dissertation Organization

This document consists of five chapters. Chapter II is a literature review, which thoroughly discusses relevant research in machine learning techniques, static heuristic methods for malware detection and classification as well as SA. Some of the research results contribute to setting performance goals for the final solution in this work, while others assist in articulating the solution's high-level impact in the context of multiple frameworks.

Chapter III describes the process followed to generate the final MaTR architectural prototype for study. It includes details gleaned from pilot tests to determine appropriate approaches to address the problem, design decisions and determine model parameters.

The experimental results in Chapter IV demonstrate the engineering advantage of the final prototype. When practical, experimental results include direct comparisons between the resulting MaTR model and results from re-accomplishing prominent tests from other research. The discussion also maps experimental findings to a common SA model.

A summary of this work follows in Chapter V highlighting the contributions and impact of this research. This concluding chapter also identifies practical areas for research extension.

1.5 Summary

This chapter introduces the research motivation and frames the fundamental problem. It also defines the major goals of this investigation. Finally, this chapter presents the research contributions of this work in terms of the malware detection and classification problems and their connections to the most common SA model.

II. Literature Review

2.1 Overview

This chapter presents a complete literature review of related and relevant research, which encompasses basic classification capabilities from artificial intelligence (AI), malware analysis techniques and finally prior research applications of machine learning algorithms to malware detection and classification. The machine learning techniques included in this review are those that dominate the AI applications in malware detection in published literature and those suggested by the success associated with preliminary experiments.

The antivirus industry has a notorious past for disagreement on general names for specific malware samples [10]. Consistent naming across vendors is important to the user community, because it simplifies communications concerning major threats. If the same threat has different names, users cannot leverage appropriate response actions. The published research in malware detection and classification using machine learning techniques utilizes a small set of feature sources and learning methods.

This chapter presents an overview of the Endsley SA model [33]. This section focuses exclusively on this SA model, because of its prevalence in the literature. Discussion emphasizes the pertinent portions of the model related to this research. A brief description of confusion matrices and associated statistics concludes this chapter.

2.2 Artificial Intelligence

This section describes in depth many popular machine learning techniques found in malware detection research. Several machine learning techniques exist that show promise in the malware detection problem, but some dominate the literature. In particular, many of the techniques have application to both continuous and discrete datasets, which is an important consideration as discrete numbers dominate the features commonly associated with computer programs. As classic antivirus signature-based solutions become overwhelmed by an exponential growth in malware, pattern recognition based solutions are gaining importance and popularity [91].

2.2.1 Discriminant Analysis.

Variants of discriminant analysis techniques include continuous multiple discriminant analysis (MDA) [28] and discrete discriminant analysis (DDA) [22]. Both techniques use common statistical methods to determine discriminant scores. Finally, a discriminant function, quite often a form of midpoint or distance measure, determines class membership for individual exemplars based on their respective discriminant scores.

2.2.1.1 Multiple Discriminant Analysis.

MDA is a generalization of the two-class linear discriminant analysis and Fisher's linear discriminant function (LDF) [28]. As linear discriminant analysis projects sample data onto a single line for the discriminant function, MDA generates multiple projections based on the number of classes and features. In many MDA applications, the first projection axis exhibits the most discriminant "value" in the dataset with subsequent projections having the same or less value in the classification.

Ideally, MDA [28] produces $(C - 1)$ linear transformations of features, where C is the number of classes, such that scores associated with objects of each class have grossly different means and minimal variances. The underlying model assumptions include multivariate normality, different class means and identical covariance matrices for all classes. The following equations follow the nomenclature of using bold English and Greek capital letters to denote matrices and bold lowercase letters for vectors.

MDA defines a within-class scatter as a measure of the variance about each class mean [28]. Equation 1 shows the formula for computing the within-class scatter matrix, $\mathbf{S}_W$, where $\mathbf{x}$ is an observation vector of features, $\mathbf{X}_c$ denotes the set of all observations of class c of C classes and the class mean vector of features is $\boldsymbol{\mu}_c$.

$$\mathbf{S}_W = \sum_{c=1}^{C} \sum_{\mathbf{x} \in \mathbf{X}_c} (\mathbf{x} - \boldsymbol{\mu}_c)(\mathbf{x} - \boldsymbol{\mu}_c)^{\mathrm{T}} \tag{1}$$

Equation 2 is the formula for calculating the total scatter matrix, $\mathbf{S}_T$, where M_c is the number of observations in class c and $\boldsymbol{\mu}_T$ is the overall class-independent mean vector of features. Since the definition of total scatter is $\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B$, where $\mathbf{S}_B$ is the between-class scatter, Equation 3 follows.

$$\mathbf{S}_T = \mathbf{S}_W + \sum_{c=1}^{C} \sum_{\mathbf{x} \in \mathbf{X}_c} M_c(\boldsymbol{\mu}_c - \boldsymbol{\mu}_T)(\boldsymbol{\mu}_c - \boldsymbol{\mu}_T)^{\mathrm{T}} \tag{2}$$

$$\mathbf{S}_B = \sum_{c=1}^{C} \sum_{\mathbf{x} \in \mathbf{X}_c} M_c(\boldsymbol{\mu}_c - \boldsymbol{\mu}_T)(\boldsymbol{\mu}_c - \boldsymbol{\mu}_T)^{\mathrm{T}} \tag{3}$$

Equation 4 simultaneously calculates all projections of vector $\mathbf{x}$ onto transformed vector $\mathbf{y}$ via the desired transformation matrix $\mathbf{W^T}$. Equations 5 and 6 are the formulas for determining the within-class and between-class scatter matrices for the *projected* data ($\tilde{\mathbf{S}}_W$ and $\tilde{\mathbf{S}}_B$ respectively) based on the scatters from the original data.

$$\mathbf{y} = \mathbf{W}^{\mathrm{T}}\mathbf{x} \tag{4}$$

$$\tilde{\mathbf{S}}_W = \mathbf{W}^{\mathrm{T}}\mathbf{S}_W\mathbf{W} \tag{5}$$

$$\tilde{\mathbf{S}}_B = \mathbf{W}^{\mathrm{T}}\mathbf{S}_B\mathbf{W} \tag{6}$$

The original goal is to maximize the between-class scatter (highly separated class means) while simultaneously minimizing the within-class scatter (minimize variance for each class). Achieving these goals is equivalent to finding $\mathbf{W}$ that maximizes $J(\mathbf{W})$ in Equation 7. Maximizing $J(\mathbf{W})$ requires $\mathbf{W}$ composed of the concatenation of the eigenvectors associated with the largest eigenvalues in the generalized eigenvalue problem in Equation 8.

$$J(\mathbf{W}) = \frac{\left|\tilde{\mathbf{S}}_B\right|}{\left|\tilde{\mathbf{S}}_W\right|} = \frac{|\mathbf{W}_\mathrm{T}\mathbf{S}_B\mathbf{W}|}{|\mathbf{W}_\mathrm{T}\mathbf{S}_W\mathbf{W}|} \tag{7}$$

$$\mathbf{S}_B\mathbf{w}_c = \lambda_c\mathbf{S}_W\mathbf{w}_c \tag{8}$$

According to Duda, et al. [28], computing the inverse of $\mathbf{S}_W$ is unnecessary and instead finding the roots of Equation 9 and then solving Equation 10 is preferable. MATLAB provides the *polyeig* command [56] for finding the roots and solving, which requires subsequent sorting operations for the resulting eigenvalues and eigenvectors.

$$|\mathbf{S}_B - \lambda_c\mathbf{S}_W| = 0 \tag{9}$$

$$(\mathbf{S}_B - \lambda_c\mathbf{S}_W)\mathbf{w}_c = 0 \tag{10}$$

After solving for $\mathbf{W}$, the correct choice for a classification function if the covariance matrices for each class are not equal is a quadratic discriminant function (QDF) (Equation 11). The value of g_c is the discriminant score for each sample for each class, where the largest score indicates the class to assign the sample. Note that Equation 11 also projects the data.

$$g_c(\mathbf{x}) = \mathbf{x}^\mathrm{T}\mathbf{W}_c\mathbf{x} + \mathbf{w_c}^\mathrm{T}\mathbf{x} + w_{c0} \tag{11}$$

In Equation 11, $\mathbf{W}_c$ (Equation 12) and $\mathbf{w}_c$ (Equation 13) are the combined projection matrices for the quadratic and linear terms and w_{c0} (Equation 14) is an offset. In these equations, $\boldsymbol{\Sigma}_c$ is the covariance matrix and $P(\omega_c)$ is the prior probability of a sample belonging to class c. Figure 2 is a sample plot of Fisher iris sepal length and sepal width data for versicolor and virginica classes and the corresponding QDF classifier. This example is an adaptation of [55].

$$\mathbf{W}_c = -\frac{1}{2}\boldsymbol{\Sigma}_c^{-1} \tag{12}$$

$$\mathbf{w}_c = \boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c \tag{13}$$

$$w_{c0} = -\frac{1}{2}\boldsymbol{\mu}_c^{\mathrm{T}}\boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c - \frac{1}{2}\ln|\boldsymbol{\Sigma}_c| + \ln P(\omega_c) \tag{14}$$

2.2.1.2 Discrete Discriminant Analysis.

Dillon and Goldstein [22] present a method of DDA that uses densities and *a priori* probabilities to classify new exemplars. In their model, a set of discrete random variables, $X_1, X_2, \ldots, X_p$, each assume a specific value from a finite set of distinct values, $s_1, s_2, \ldots, s_p$. Here x denotes a specific realization of X. The set of all distinct values is given by the product $\prod_{i=1}^{p} s_i$. The discriminant score for an exemplar is given by Equation 15. In Equation 15, $f_i(x)$ is the class conditional density for class i and P_i is the *a priori* probability for class i (see Equation 16) with n_i number of class i samples and n total number of samples.

$$g_i(x) = P_i f_i(x) \tag{15}$$

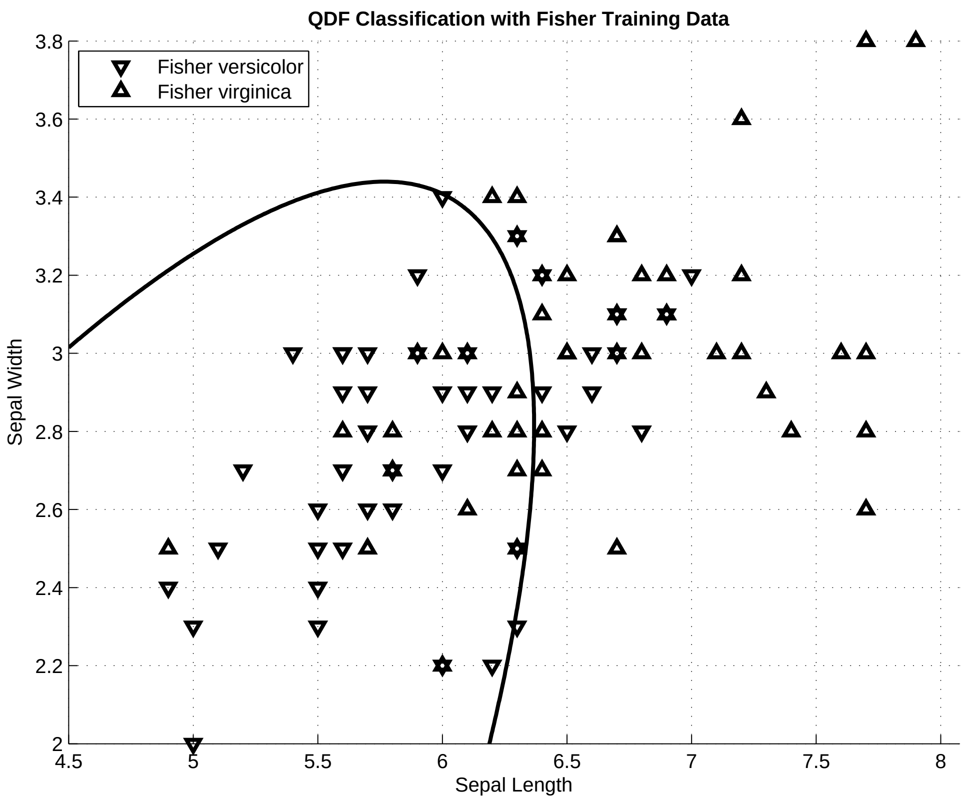

Figure 2. Sample MDA with QDF decision boundary (adapted from [55]).

$$P_i = \frac{n_i}{n} \tag{16}$$

Equation 17 shows the estimate for class conditional density, where $n_i(x)$ is the number of observations from class i with characteristic x and n_i is the number of observations from class i. The simplified estimate for the discriminant score, $\hat{g}_i(x)$, is Equation 18. Finally, the discriminant function assigns class membership to the highest discriminant score. In the event of equal scores, the assignment is random among the classes associated with the equality.

$$\hat{f}_i(x) = \frac{n_i(x)}{n_i} \tag{17}$$

$$\hat{g}_i(x) = \frac{n_i(x)}{n} \tag{18}$$

This method performs poorly when the feature set enumeration does not clearly separate the classes as the sample data of cancer observations in New York of males and females from [82, 83] shown in Table 3 suggests. Based on the table, the model assigns all new exemplars with feature $x_1 = 1$ to Class 1, because Class 1 has a higher discriminant score than Class 2, but only about half of the samples with feature $x_1 = 1$ actually belong to Class 1. Nearly half of exemplars with the trait $x_1 = 1$ are misclassifications leading to high error rates. The ideal cases in this example are when features $x_1 = 2$ and $x_1 = 3$, which demonstrate mutually exclusivity between classes. In spite of its drawbacks, this model has inherent advantages, such as the ability to trivially handle the exclusive `or` (aka `xor`) problem.

2.2.2 Feedforward Neural Networks.

A feedforward neural network (FNN) is a model motivated by models of neurons in the brain. Figure 3 shows the biological inspiration for neural networks. In a greatly simplified explanation of this theory, a neuron sends signals down its axon which may

Table 3. Discriminant scores for cancer data using DDA.

Cell	Class 1 (Males)		Class 2 (Females)	
$\mathbf{x_1}$	**Observed**	$\hat{\mathbf{g}}_\mathbf{1}$	**Observed**	$\hat{\mathbf{g}}_\mathbf{2}$
0 (Lung)	7,355	0.268	4,831	0.176
1 (Melanoma)	1,104	0.040	964	0.035
2 (Ovarian)	0	0	1,563	0.057
3 (Prostate)	9,986	0.364	0	0
4 (Stomach)	1,014	0.037	618	0.023

pass these signals via a synaptic connection with a receiving neuron's dendrites [38]. The theory also describes how data processing takes place in the neuron and synaptic connections serve to share this information with other neurons.

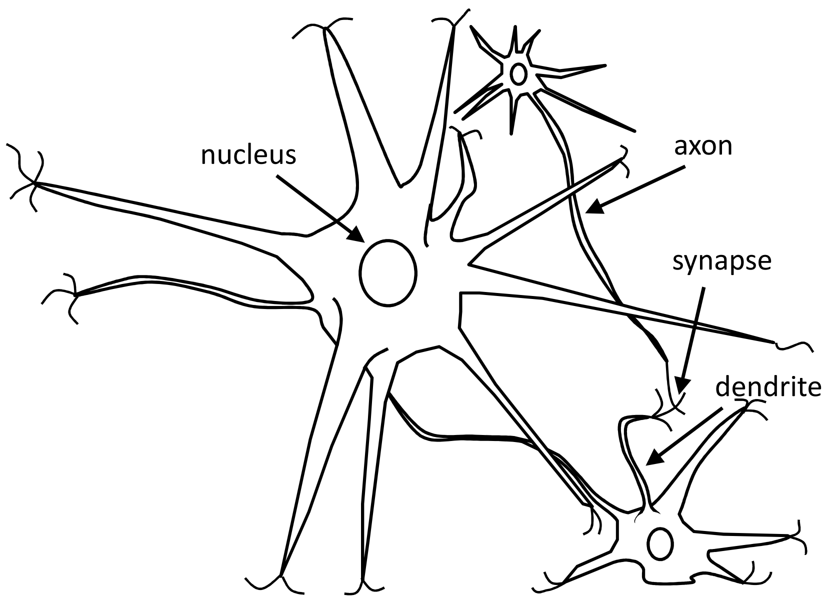

Figure 3. Biological inspiration for neural networks.

According to Hebb [38], when an axon of cell A often excites or contributes to the firing of another cell B, internal processes cause cell A to become more efficient at causing cell B to fire. His observation has become the foundation for neural network model learning. This observation is evident in a basic form of neural network called the perceptron model.

Rosenblatt's [73] perceptron model of a neuron forms the basis of a multilevel graph structure that supports machine learning. Figure 4 shows a diagram of Rosenblatt's simple perceptron model. Rosenblatt's perceptron provides a mathematical model that demonstrates a simple processing unit that receives inputs, performs processing, and produces outputs for the network. In addition to inputs and outputs, the model also contains potential and an activation function.

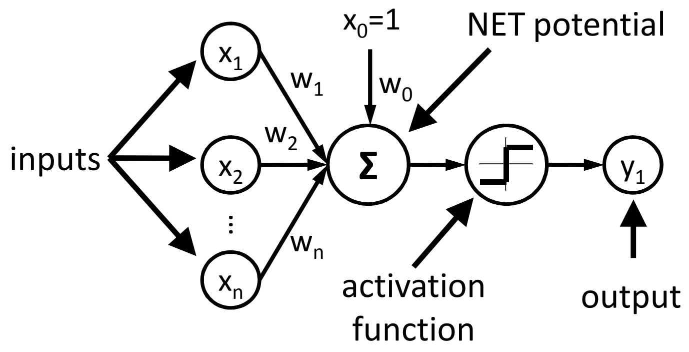

Figure 4. Rosenblatt's simple perceptron model [73].

In the perceptron model above, x_n represents the presynaptic stimulus, w_n is the synapse stimulus concentration (aka "weights"), and $w_n x_n$ demonstrates the postsynaptic action potential for each dendrite. Individual weights demonstrate Hebbian learning as the weight magnitudes directly induce whether or not the forward neuron fires. Thus, the action potential of all of a neuron's dendrites is the summation of the appropriate products (Equation 19). The bias term, x_0, is necessary to allow the network function result to shift away from the origin.

$$NET = \sum_{i=0}^{n} w_i x_i \tag{19}$$

The perceptron activation function determines whether the neuron axon outputs fire or not. In Rosenblatt's simple perceptron, the activation function is a hard limiter as shown in Equation 20. Common activation functions include the sigmoid and hyperbolic tangent functions [28] shown in Equations 21 and 22 respectively. These functions have the advantage of allowing the neural network to generate nonlinear functions as shown in Figure 5.

$$y(x) = \begin{cases} 1 & \text{if } NET > 0 \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

$$y(x) = \frac{1}{1 + e^{-x}} \tag{21}$$

$$y(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{22}$$

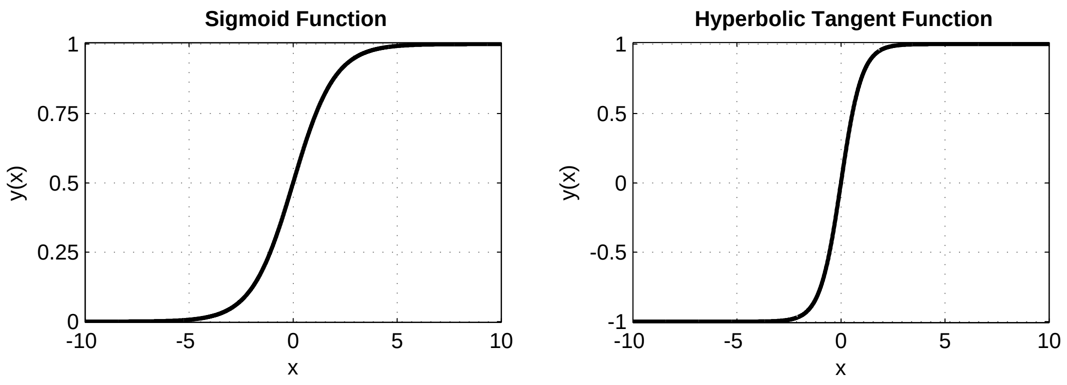

Figure 5. Plots of sigmoid and hyperbolic tangent activation functions.

The perceptron model provides a mechanism for learning by adjusting the various network weights. Although numerous perceptron-based neural network implementations exist, this discussion focuses on only the FNN. The FNN places additional connectivity limits from lower layers to higher layers only, hence generating a “feedforward”-only network. In order to train weight values, backpropagation of some error representation is necessary. The most common applications for the FNN are regression and classification.

Duda, et al. identify several methods of ensuring faster backpropagation convergence of the FNN [28]. They recommend activation functions that are nonlinear, saturating, continuous and monotonic. Standardization of input features to have zero mean and unit variance prevents the network from overemphasizing larger magnitude features. Target values should not exceed the range of possible network output. For example, a network with a hyperbolic tangent sigmoid activation function should not attempt to train to target values exceeding magnitude 1.0, because the network is incapable of reaching the target values causing obvious error.

Duda, et al. [28] recommend choosing the number of hidden units to cause the number of weights in the network not to exceed $\frac{n}{10}$, where n is the number of training samples. They indicate that although some have enjoyed success with more hidden nodes, a "principled approach" is to initially use more hidden nodes, and then prune weights to the point of elimination for insignificant associations. The authors also describe techniques for determining learning rates and the potential advantages of momentum.

Kivinen and Warmuth [45] discuss matching network loss and activation functions to allow faster convergence in neural networks with multidimensional outputs and no hidden layers (generalized linear regression). They also suggest that the optimal network learning rate, η, also depends on the loss function. They identify the relative entropy function (see Equation 23) as the appropriate matching loss function for the sigmoid activation (or transfer) function.

$$L_\sigma(y, \hat{y}) = \sum_{j=1}^{k} y_j ln \frac{y_j}{\hat{y}_j} \tag{23}$$

Figure 6 shows a sample multilayer FNN with perceptron nodes. This particular sample has input, (single) hidden and output layers. Multiple hidden layers may exist, but some claim the use of a single hidden layer is sufficient for approximating any continuous multivariate function based on Kolmogorov's theorem although Girosi and Poggio disagree [36]. They indicate the Kolmogorov network is not a parameterized representation as neural networks and the inner functions in Kolmogorov's theorem are not smooth causing the approximated function to not be smooth, which makes generalization more difficult. They also claim that Kolmogorov's theorem "cannot be used by itself in any constructive way in the context of networks for training" [36].

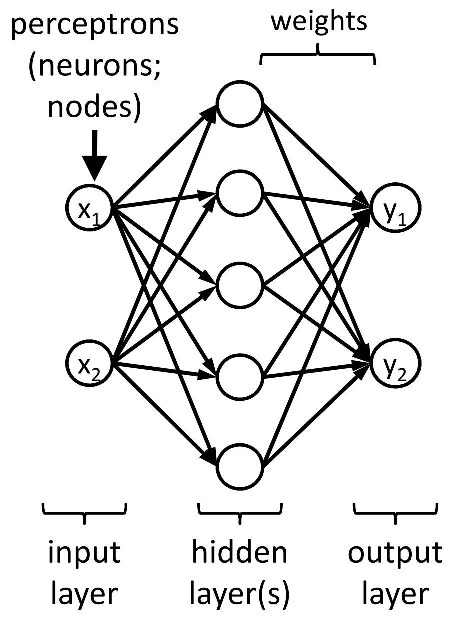

Figure 6. Sample neural network with hidden layer.

The FNN model is not without limitations. The nodes in the model do not necessarily represent any specific information at all, which makes interpretability and reuse of portions of the network impossible. This representation also makes intuitive validation of FNN performance difficult, because the only predictable model output is the final result. If environment stimuli change, attributing potentially degraded performance to any specific cause is difficult and requires periodic retraining and analysis.

For example, Figure 7 shows a neural network that takes two inputs (x_i), has five hidden layer nodes (n_j), and a single output (y_1). When fully trained, the operator cannot deduce interpretable information from any particular node regardless of overall model optimality. For instance, a trained network to identify red circles inevitably does not represent "redness" or "circle-ness" at any particular node, but rather only represents "(red circle)-ness" at node y_1. The conceptual representation for "redness" may be local to a specific node although more likely spread out over a larger group of neurons possibly encompassing the entire model. The lack of fine-grain informational representation prohibits any other network from utilizing a trained *portion* of the network even if the network performance is optimal. Reuse of a portion of a network requires complete retraining, a wasteful practice.

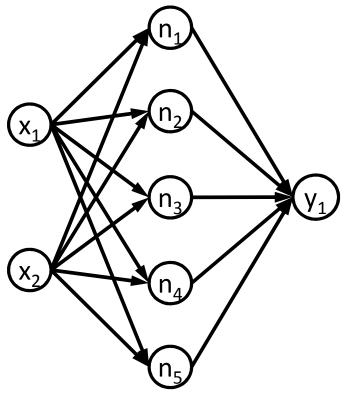

Figure 7. Sample neural network for context discussion.

The network in Figure 7 is also useful to discuss the likely overuse of weighted associations in the FNN. For example, if n_1 and n_2 adequately represent "redness" and "circle-ness", then the cross associations between x_1 and n_2 and also x_2 and n_1 are most likely unnecessary as features contributing to one characteristic likely are independent of the other. Duda, et al. recommend a pruning solution to address this scenario [28], but effective pruning is impossible if the network coalesces redness and circle-ness into the same set of nodes. The number of weights and the number of training samples most commonly influence network structure decisions, which further emphasizes this problem.

2.2.3 Decision Trees.

Unlike the previous techniques, decision tree (DT) models are more intuitive, bearing strong resemblance to the classic “20-questions” game. As a more natural classifier for nominal data, the root “node” of a directed tree structure *splits* the data into smaller subsets with a question whose response space is *exhaustive* and *exclusive* [28]. Determining precisely which features to use in splitting is a significant concern with this model and the DT can use the same feature for multiple splitting decisions. A subset is *pure* if it only contains samples from a single class.

Reaching a *leaf* node results in a classification, while non-leaf nodes present a series of splitting questions. In the event of a pure subset, the decision to make a node a leaf is trivial, but impure subsets require a judgment between accepting error or further tree expansion. The decision rules of large trees are often complex and require tree reduction to simplify.

Duda, et al. identify interpretability as a major benefit of DTs over neural networks [28]. The role of every node in a properly reduced tree is much clearer and the decisions reached at the leaf nodes are more obvious. As previously stated, the only node function in an FNN that is obvious is at the output layer and the information represented at other nodes is highly questionable.

Figure 8 shows a DT that classifies red circles. In this trivial example, the root queries whether the sample is red and the right subtree queries whether the sample is a circle. The leaf nodes assign distinct classes of membership to each sample. Each node has a clear interpretation unlike the previous FNN model. In a similar application, the FNN may represent redness in a single node although more likely across several nodes—even combining concepts like redness and circle-ness in the same nodes. Such connectedness prohibits substantial pruning as most of the network represents most of the necessary concepts.

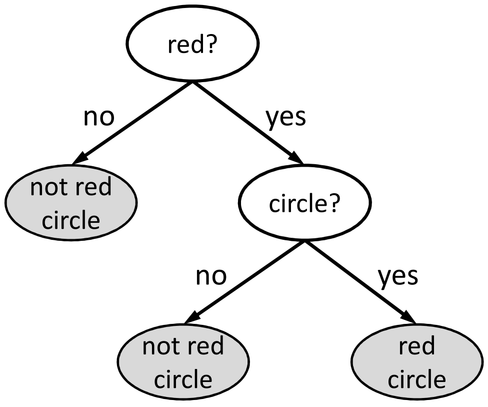

Figure 8. Sample decision tree for red circles.

Duda, et al. present a generic tree-growing methodology called the classification and regression tree (CART) [28]. They list six design considerations to address when building a DT:

- branching factor to use for each node,
- feature to use for splitting a node,
- conditions for deciding a node should be a leaf,
- pruning techniques,
- determining class for impure leaf nodes and
- handling missing or noisy data.

The *branching factor* of a node is the number of edges leaving a non-leaf node. Many applications use binary trees (i.e., a branching factor of 2) [57], because they are both efficient implementations and capable of representing any complex decision logic.

Determining which features to use for splitting is a practical application of Occam's razor, a classic argument for simple solutions [21]. The most concise DT with the same decision-making results is preferable to a grossly enlarged tree that adds complexity and suffers from poorer performance. A concise DT has generally smaller overall distances from the root to the leaf nodes, which implies that a reasonable goal of each decision point is to generate descendants with maximal purity, or minimal *impurity.* Theoretically, these decision points may utilize multiple features and such trees are *polythetic*; DTs that restrict themselves to single features for each decision point are *monothetic*. MATLAB's classification DT implementation considers a random selection of $\sqrt{n}$ features from n total features for consideration as the monothetic cut variable [57].

Various impurity measures exist, such as entropy, variance, Gini, and misclassification impurities [28]. In all of these, the impurity of a node N, denoted $i(N)$, is zero if each subset contains samples from only one class. Obviously, the classes for each subset are different. In the case where $i(N) \neq 0$, the impurity is positive with the highest value occurring when different classes occurring with uniform density in a subset. Incorporating cost functions can shift decision boundaries as with other classification techniques.

Equation 24 is a function that calculates the change in impurity of a feature and corresponding value for the decision point at node N [28]. In this equation, N_L is the left node of the split operation (i.e., descendant), P_L is the percentage of samples that N_L contains and $i(N_L)$ is the calculated impurity for the N_L. The right side variables follow a similar convention. The best possible decision using this greedy method is to split on the feature and value that produces the highest $\Delta i(N)$. Making an optimal local decision, however, does not guarantee an optimal global solution, a common drawback of greedy algorithms.

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \tag{24}$$

The critical performance decision is defining the conditions identified for declaring a node a leaf. Surprisingly, this decision is even more important than the impurity function chosen in regards to classifier accuracy according to Duda, et al. [28]. Training DTs is quite similar to neural networks, because continual node splitting leads to overfitting of the data and the typical loss of generalization while stopping too soon is essentially underfitting. Techniques for determining when to stop include threshold comparisons of split impurity reduction or establishing a minimum number of samples a node can represent. Other statistical methods exist as well, such as testing if Δi is significantly different from zero, hypothesis testing, and confidence level testing.

Another approach to building DTs is to first build an exhaustive tree and then progressively *prune* away leaf nodes, or *merge* them, until the leaf nodes meet certain criteria. Instead of splitting nodes to build a DT, pruning has a major advantage of immunity to the *horizon effect*, where splitting decisions do not consider future beneficial splits of descendant nodes because they do not exist yet. As a complete DT is by definition overtrained, pruning is necessary for generalization.

While assigning class labels to leaf nodes with zero impurity is trivial, leaf nodes with positive impurity are not difficult to assign either. Assigning class membership based on the class of preponderance results in lower classifier error. Expanding the tree to reduce this error results in undesirable overfitting.

Duda, et al. indicate that a major limitation of DTs is computational complexity [28]. Given several generic assumptions, for n samples and d features, the training performance is $O(dn^2 \log n)$ with a classification performance of $O(\log n)$. The classification performance is superior, but the dismal training performance is often prohibitive for large numbers of samples or features.

2.3 Malware Naming and Analysis Techniques

Malware naming and analysis is more art than science [10]. The malware research community consists of numerous disparate organizations and commercial companies. Overall, this community does not suffer overly adverse effects for not settling on rigid, industry-wide standards for naming analysis. The potential losses can easily outweigh the advantages for this industry. While many antivirus analysts may agree that a specific sample program is malicious, they often disagree on malware names and the best descriptions for the sample. In many cases, they come to their conclusions by entirely different analysis processes. This section describes the difficulties the malware research community has with naming conventions and common malware analysis techniques.

2.3.1 Naming Convention Problem.

The antivirus research community has demonstrated difficulty in naming newly discovered malware since its beginning three decades ago. With little motivation to work together to define a consistent and useful naming scheme, the status quo leads to confusion between vendors, Information and Communication Technologies (ICT) staffs, and the general public. Although collaborations among antivirus researchers are rare (usually only during major virus outbreaks), a universally accepted naming convention expedites information sharing for the good of end users. Such a standard simply does not exist even after repeated efforts by industry and non-profit organizations.

Each vendor has their own local policy that dictates their naming conventions. Conceding to an industry-wide naming scheme requires the concession of a degree of autonomy and likely imposes more diligent "busy" work on company analysts.

Through collaboration among vendors and non-profit organizations a less biased scheme emerged, but universal acceptance and adherence does not exist.

Some non-profit groups propose different standards, but most omit multiple common malware types, such as backdoors and downloaders [64]. Arguably, the most influential standard in the antivirus community comes from the Computer Antivirus Researcher's Organization (CARO) [10, 91]. Mitre's Common Malware Enumeration (CME) attempts to address the differences in antivirus company naming by establishing neutral identification numbers for vendors to reference in addition to their own assigned names [64]. Microsoft has also begun touting its malware naming scheme, which appears to be a simplified version of the CARO standard [63].

2.3.1.1 CARO Naming Standard.

CARO is an informal organization of antivirus researchers from multiple corporations and academia who collaborate on limited malware research issues. One particular focus area of CARO is the establishment of a naming standard for new malware [10]. The general form of malware names under the CARO Malware Naming Scheme is the following

[type://][platform/]family[.group][.length].variant[modifiers][!comment]

where the items in brackets are optional. As evident, the only mandatory fields are *family* and *variant*.

Much of the emphasis in the CARO standard addresses virus families, but the guidance is not definitive. Virus families are helpful to researchers, because they provide a crude measure of similarity between viruses. Similar viruses likely have similar infection capabilities, defenses, and require similar disinfection techniques and tools [10]. Reuse of previous work in any form is highly desirable and accelerates production and responsiveness.

One problem with the CARO naming scheme is that antivirus vendors are under no firm requirement to adhere to the standard [10]. Participation is strictly voluntary and as a result, no vendor product is fully compliant with the standard. Furthermore, some researchers criticize the standard [10] for various reasons, but many reluctantly abide by the overarching spirit of the naming scheme.

Other problems are more semantic in nature. For instance, *Win32/MyDoom.BQ* and *Win32/MyDoom.ED* are syntactically correct names based on the CARO standard [10], but different vendors may use these different names based on local policies, which results in a misnomer or at least an inconsistency. In most cases, antivirus companies refuse to change their assigned names after publishing the information to the media or online databases, because of the additional confusion caused and potential loss of credibility.

CARO presents the following prioritized list of malware types and definitions [10]:

1. **Virus**: “program (or a set of programs) that can replicate itself recursively”
2. **Dropper**: “malware that does not replicate itself but which releases self-replicating malware”
3. **Intended**: “malware written with the obvious intent to write a virus but which fails to replicate”
4. **Trojan**: “malware that does not even try to replicate itself but which performs some intentionally destructive action, without correctly warning the user”
5. **PWS**: program whose primary purpose is to “steal passwords”
6. **Dialer**: “program that installs itself in the chain of programs invoked when the computer is establishing a dial-up connection. The purpose of such a program is to force the connection to the Internet to go through a particular premium phone number.”

7. **Backdoor**: "program that allows access to the machine on which it has been installed, access that circumvents the legitimate login authentication procedures for that machine"

8. **Exploit**: "way of bypassing the security of a program or an operating system"

9. **Tool**: "program that is not dangerous to the user who runs it, but that can be used to produce malicious programs or to perform malicious actions"; examples include virus construction kits, password cracking and other attacker tools

10. **Garbage**: "various programs that do not perform any meaningful action (usually due to bugs) and do not even try to be viruses (or they would be classified as 'intended')"

Leveraging subclass information from the antivirus community is critical for incorporating their expert findings into academic research. The CARO standard attempts to address some glaring issues, such as how to assign a sample to a specific malware class given that it exhibits characteristics common to multiple *types*. The above list is in order of precedence of "worst types" with *virus* being the worst. As a simple example, if a program steals passwords and replicates, its class is *virus* according to a strict interpretation of the standard.

The above list is also the only allowable *types* of malware according to CARO. They do not include several commonly accepted categories of malware, such as *worms*, *keyloggers*, *spam*, *adware*, *spyware*, and *phishing scams*. Bontchev [10] indicates that the *worm* category is not in the standard, because antivirus researchers do not collectively agree on a precise definition for it. To avoid ambiguity, the CARO standard omits the category and instead advises researchers to include appropriate information in the *comment* field of the scheme. Other types are omitted, because of inadequate definitions and CARO's interpretation of applicability to antivirus scanners. The standard also refers to the popular *keyloggers* as *pws*, or password stealers, but such programs can steal more information than just passwords. In time, CARO may add more types to the acceptable list.

As a practical example, the Symantec named *W32.Wargbot* is a worm that opens an Internet Relay Chat (IRC) backdoor on a system [67]. The analysis report continues calling the same specimen a Trojan also. According to the CARO standard, the class is *virus*, because worms replicate and the standard does not currently allow for the more specific category *worm*.

Another interesting point is that the Symantec name for *W32.Wargbot* does not include the *type* field in the name at all. The name *W32.Wargbot* is likely what their scanner returns when the virus is found—not necessarily the full name of the virus as defined at Symantec. CARO only requires scanners to report the family name when it detects malware. Vendors are free to omit the malware type even though this information is likely the most important to antivirus customers. Customers must determine the malware type themselves by scouring antivirus companies' websites.

The CARO naming scheme precedence may simplify the naming for the antivirus research community internally, but not for customers. Users of antivirus solutions would obtain more information by knowing that a particular infection involved a *backdoor* or *pws* (or *keylogger*) rather than the generic term *virus*. The former directly implies an obvious sequence of remediation steps to execute immediately after disinfection to ensure confidentiality; the latter does not directly imply any necessary follow-up action other than disinfect and remove.

The CARO naming scheme *type* is also a mixture of malware propagation methods and payloads, but the standard calls for a single, overall *type*. If vendors do not provide the additional information in the *comment* field of the name, end users must search elsewhere for relevant threat information. Propagation (or replication) methods describe how malware spreads across systems and networks, not other malware functionality. Swain [86] identifies three major types of malware for Symantec, Trojans, worms and viruses. Trojans are non-replicating [86, 91], while worms and viruses propagate [64, 86, 91] with the distinction being that worms do not require user interaction to spread [91].

Malware "payload" refers to the functionality of the malware in terms of adversary utility or victim annoyance. Szor [91] describes several different types of payloads. While one identified payload, "no payload", is relatively innocuous except for its propagation, other payloads can overwrite data, destroy hardware, deny service, steal data or provide unauthorized remote access to an adversary via a backdoor.

2.3.1.2 Common Malware Enumeration.

Mitre establishes a neutral identification number for antivirus vendors to include in their malware descriptions to alleviate confusion caused by multiple vendor names to describe the same malware artifact [64]. The CME intent is to address "pandemic" malware threats by issuing common identifiers to those limited samples that experts project to have widespread impact. With a common label for these samples, infected organizations can better understand the situation and execute the appropriate response actions.

Mitre also offers definitions for the following malware categories in the context of the CME [64]:

1. **Virus**: "a program that infects a computer by attaching itself to another program, and propagating itself when that program is executed"

2. **Worm**: "a computer program that can make copies of itself and spread itself through connected systems and using up [sic] resources in affected computers or causing other damage"

3. **Trojan**: "computer code that does something that is not expected by the executor of the code"

Although the above definitions are reasonable to experts in the field as they loosely correlate to the industry "standard" of propagation methods, they are grossly incomplete by themselves. The lexicon does not address malware payloads, such as *backdoors*, *keyloggers*, and *rootkits*. The manner of malware propagation implies little about its functionality. An annoying *virus* that pops up a window every morning after login saying, "Happy Birthday"—albeit highly annoying—is not as significant a threat as a *backdoor* or *keylogger*. Although not its primary motivation, a primary limitation of this lexicon as a standard is the lack of payload identification.

The now defunct CME list did not last long even though it improved coordination during widespread outbreaks. In total, Mitre assigned only 39 identifiers over a two year period and has since archived the CME list and diverted their efforts to secure software development practices [64]. They cite that in the end of 2006, malware attack patterns changed to target individual users more than global threats and the service is no longer necessary [64].

In the previous example of the Symantec named *W32.Wargbot*, the CME identification number is 762 (referred to as "CME-762"). Table 4 shows the different names that antivirus vendors use when referring to CME-762 [64]. The table entries reveal that no vendor that explicitly included the *type* field from the CARO Malware Naming Scheme followed the standard, because the malware *type* should be *virus* according to the Symantec description [67]. Familiarity with the industry naming standards may lead one to believe that several vendors imply *virus* as the *type* by including the *platform* without any other explicit *type* information (*virus* is the default in the absence of other *type* information). Collectively, the antivirus vendors assign all three propagation methods to this single sample, which is a logical conflict with the CARO standard as a *virus* cannot also be a *Trojan* [10]. The malware families identified also demonstrate a lack of continuity across vendors with few having even similar *family* names. A few vendors categorized the malware *type* as a *backdoor*, which is likely a more useful description for customers.

Table 4. Antivirus vendors names for CME-762.

Vendor	Name
Avira	Worm/IRCBot.9374
Authentium	W32/Ircbot.TT
CA	Win32/Cuebot.K!Worm
ClamAV	Trojan.IRCBot-690
ESET	Win32/IRCBot.OO
Fortinet	W32/Graweg.A!tr.bdr
Grisoft	BackDoor.Generic3.GBB!CME-762
Kaspersky	Backdoor.Win32.IRCBot.st
Microsoft	backdoor:Win32/Graweg.B
McAfee	IRC-Mocbot!MS06-040
Panda	W32/Oscarbot.KD
Sophos	W32/Cuebot-M
Symantec	W32.Wargbot
Trend Micro	WORM_IRCBOT.JK

2.3.2 Analysis Techniques.

Malware categorization is a product of functional analysis of samples. Categorization often requires visibility into the operating system (OS) application programmer interface (API) calls of the program. In order to accomplish certain tasks, such as interface with the network or file system, a program usually calls the respective OS API. In general, the two major approaches for obtaining this information are static analysis (aka, whitebox or clearbox testing) and dynamic analysis (aka, blackbox testing). Static analysis does not require central processing unit (CPU) emulation [87, 91]. Dynamic analysis requires CPU emulation or actual runtime observation. Fortunately, nothing prohibits mixing between the two techniques and many analysts use a hybrid of the techniques or whichever method they believe is more advantageous to the specific task at hand.

2.3.2.1 Static Analysis.

Static analysis takes an inside-out approach focusing on a thorough examination of the program instruction disassembly to determine its capabilities. A more universal example of this approach is to determine the functionality of an automobile by examining its parts (i.e., classic reverse engineering). Static analysis of malware normally requires the use of a debugger, but purists might argue that only a disassembler is necessary. Although a slight misnomer, most analysts performing a static analysis "step", or execute the program one instruction at a time, through certain code sections to verify functionality or to examine a particular defense. Technically speaking, this "stepping" is execution of the program, which is the approach of dynamic analysis. The primary difference is that the analyst is examining the internal instructions to determine the function, not the external behavior.

Malware researchers conducting static analysis benefit greatly from taking detailed notes concerning how they unpacked packed executables and disinfection techniques [91]. Most of the time malware authors reuse the same packing tools and do not make new packers, which simplifies the process for analysts. Also, cleaning techniques are similar for samples from the same malware family [10, 91]. Keeping good records prevents analysts from performing tedious steps again and allows them to refine their standard analysis approach.

Normally, the first step is "unpacking" of the program's real instructions into memory, which may require extended periods of time during static analysis if the packer is new or advanced. Antivirus researchers normally make a modified version of the program that is not packed. Nearly all further analysis uses the unpacked version of the malware instead of the original sample, because reinitializing the program in a debugger is faster and justifies the few minutes taken to modify the executable.

For many packers, such as the popular packer Ultimate Packer for eXecutables (UPX) and its many variants, the unpacking process is almost trivial taking an expert analyst just seconds to unpack. One reason a malware author would use such a tool even if it only stops a human expert for a few seconds is to prevent an automated antivirus tool from making an easy heuristic signature match, instead of forcing analysts to manually generate a new signature. According to statistics from ShadowServer [79], a non-profit organization that analyzes and reports on malware and botnets, UPX variants account for over 50% of all the packers used in the samples observed from June 2010 to June 2011.

One of the most difficult tasks in static analysis is determining that the disassembly is correct [65]. Many researchers overlook this, assuming that the true disassembly is available and then performing analysis on a "pristine" disassembly. Research making this assumption includes [14, 44, 85] and many others described in a recent survey paper [80]. Methods to obtain a clear disassembly include manual verification and performing an execution trace in a debugger—neither of which are conducive to automated solutions by attacker design. Execution traces can be quite slow especially when facing malware anti-emulation techniques [91]. Today the unpacking process is simple, because most available packers are not too advanced [91]. Nothing prohibits this situation from changing at a moment's notice, which would cause ripple effects across the research community. Packers can encrypt portions of the executable until needed while simultaneously shuffling them around in memory. Disabling this defense is also not trivial depending on the structural implementation the packer uses.

After the program is completely unpacked, the process of identifying the program's functionality begins. This portion of the process becomes extremely "artistic" with each analyst examining the program based on their own personal process, experience, and intuition. For example, some prefer to read through the instructions periodically making notes and rarely execute any instructions. This particular technique is effective, but analysts must keep mental track of state data, which is difficult in large programs. Others scan the program for interesting instruction sequences, set breakpoints and execute the program until it pauses at the breakpoint, where they can examine the program state.

One major analyst goal is to find the OS API calls and determine the program functionality based on sequencing and presence. A program that does not invoke network API calls cannot function as a *backdoor*. A program that has API calls to monitor keystrokes and to write to the disk may possibly be a *keylogger*, or *pws* according to CARO [10]. By setting breakpoints on API calls to initialize network sockets, the analyst can quickly observe the details of the network connection, including the remote host name or Internet protocol (IP) address. All of these observations are critical pieces of information in determining the malware *type*.

Analysts can observe all pertinent functionality of a fully unpacked program assuming they have all of its components. If the analyst is missing a key, custom dynamic-link library (DLL), the analyst cannot observe or run the entire program. This problem can cripple dynamic analysis, because necessary program code is missing. The missing code may contain significant portions of the program's functionality. In the previous examples, if the DLL contains the keystroke monitoring API calls, but analysts do not have a missing DLL, they may categorize the sample incorrectly. However, analysts can conduct a partial static analysis of the functionality present in the available program components, but this approach makes bold assumptions.

Static analysis generally requires less special hardware than dynamic analysis, because ideally the analyst does not "execute" the code. As described earlier, analysts may step through portions of the code, but this technique may require access to the same special hardware associated with dynamic analysis depending on the functionality of the code.

Static analysis requires discipline and skill in order to be effective. Simply skimming the disassembly is not adequate, because analysts have no way of knowing that they are seeing the true disassembly [65]. Malware defenses include subtle tricks to prevent antivirus researchers from “cheating” by taking too many shortcuts and keep the “game” interesting. On the other hand, a team of disciplined and skilled analysts theoretically can reverse engineer any software program, because the program must expose the true instructions prior to the computer attempting to execute them. Otherwise, the program simply crashes and does nothing except embarrass the author and possibly annoy the victim.

2.3.2.2 Dynamic Analysis.

Dynamic analysis is the complementary technique to static analysis. Instead of examining the internals of the program instructions, it does not even consider such a low-level detail. Dynamic analysis focuses on the sample’s interaction with its external environment for categorization. If the program exhibits a particular behavior consistent with a specific malware type, then it likely belongs to that category.

This blind assumption is arguably the greatest weakness of dynamic analysis. While static analysis’ main weakness is the inability for analysts to know they are examining the correct and entire disassembly [65], the dynamic analyst can never know if they observed the sample for a sufficient duration. With so many malware samples exhibiting behavior of waiting until a certain date (à la Michelangelo virus), this assumption is simply not true.

Special environments might alleviate this concern by advancing the virtual clock at a greatly accelerated rate, but they assume samples cannot detect this activity by observing how much time elapsed between processor cycles (or even OS context switches). Tactically, if a sample is advanced enough to detect this type of obvious analytical behavior, it will also choose to mask its true function prolonging its time in the wild. Ultimately, clock acceleration does not guarantee that the analyst witnesses the true capability of a particular sample.

Dynamic analysis generally requires more resources than static analysis, but different implementations exist depending on the level of fidelity required. Simulated environments can provide pseudo-infrastructure to allow the malware to execute. "Sandboxing" also provides self-contained environments that encourage the sample to execute.

Simulated Environment. To fully analyze malware in action, an isolated, simulated network environment is often necessary. One such implementation involves having actual hardware or virtual representations set up in a realistic network configuration. This environment must allow for simple traffic capture and analysis as this becomes a critical component of the entire analysis process for monitoring network traffic. An IDS is beneficial for testing samples against known signatures when they operate on the network. Some malware variants attempt to exploit critical network services as part of their propagation or payload, such as worms.

This environment is a simplified simulation of enterprise network resources. Major internal services include a domain name server (DNS), domain controller, and mail server. Other network components that malware may target are web and file servers. The environment can simulate all services virtually on one system, if a single system can handle the required bandwidth.

Traffic analysis with a network sniffer is a critical component of the dynamic analysis environment. It allows the analyst to identify quickly any domain names the malware attempts to resolve and nonstandard protocol use. Some malware attempts to resolve common websites like Yahoo and Google in an attempt to verify network connectivity before attempting to contact its own command and control (C2) server. Many of these network anomalies are decent malware indicators and traffic analysis techniques help to quickly identify them.

The major reason to simulate this infrastructure is to observe what the malware will do in a realistic environment. If the malware sample attempts to resolve a domain name and establish a web connection to its C2 server, the environment needs to include at least a mock DNS and a mock web server. The analyst can add a resolving entry for the requested domain name pointing to the mock web server. When the malware connects to the mock web server to request a page, the observer can analyze the network traffic for anomalous communication patterns for potential IDS signatures. After deploying any potential signatures, the company can proactively scan its networks for instances of the malware.

Another noteworthy point is the fact that malware families tend to have strong similarities possibly including communication protocols. Developing custom signatures for previously unknown malicious artifacts may not only identify the same artifact but also close variants to it possibly within the same family. Furthermore, in the case of polymorphic viruses, which change themselves so dramatically during propagation that antivirus signatures are difficult to generate, the underlying functionality and communication methods may be quite similar allowing for signature re-use. In these cases, a custom IDS signature proves absolutely essential.

The analyst stations must also employ some protective measures as well. Nearly all antivirus researchers highly recommend examining malware in a virtual environment that enables quick restoration of the system to a pristine state [91]. After analyzing a specific sample, the analyst should revert to a virtual image or re-image the system entirely and the virtual environment greatly simplifies this process.

In addition to protecting the analyst station, the system also needs tools that provide visibility into low-level malware actions, such as the free program *Process Monitor* [75]. This tool logs all registry and file system accesses. Other tools, such as *netstat*, dump all network connection data and the processes associated with them. Discrepancies in the locally observed data on the analyst station and the network sniffer are possible indications that malware may be present.

Sandboxing. Sandboxing has quickly become a phenomenon, even though it does not address all issues of traditional dynamic analysis. Several companies now develop sandboxing technologies with various options for online and local analysis. Online analysis requires users to upload the files in question to a website and the analysis engine returns a standardized report normally within a set runtime duration specified by the user. Local analysis allows the user to run the same analysis engine on a user network.

Sandboxing has some advantages over the simulated environment described above. Sandboxes employ a technique called API "hooking", which allows the environment to intercept API calls. As such, they have the capability to report all "hooked" API calls that the sample attempts. These environments allow the malware to gain access to real external services, like DNS and other Internet services, which can be dangerous. Normally, the runtime duration allowed by the online analysis is quite brief, which presents an obvious limitation. Examples of popular commercial sandboxing solutions include Norman Sandbox [68] and Sunbelt's CWSandbox [84].

2.3.2.3 Antivirus Heuristic Scans.

Antivirus heuristic scans are certainly worthy of note, but limited in effectiveness for large enterprises specifically targeted by threats. Heuristic scanners key on information typical of a malware type and provide an alert if the file looks suspicious enough. Antivirus vendors closely track statistics based on specific program characteristics, such as file size, evidence of obfuscation, the number of program sections, section alignment, executables running as services, and the import of specific libraries [72]. They may also have signatures that represent general code sequences to accomplish a particular task, such as Symantec's Bloodhound technology [87]. With generic enough signatures, they deduce sequences of suspicious code.

They use this information as features for their heuristic scanners. Unfortunately, these scanners tend to ship with either insensitive settings or malware authors test their newest products against them before infecting victims. From a practical standpoint, they are not overly effective for finding new or modified malware [14, 15]. Normally heuristic scanners cannot be run independently of the main product signature-based engine.

2.3.2.4 Multiscans.

Multiscan services are popular as well, because they offer antivirus scan results from numerous vendor tools at the same time. The primary limitation of such tools is the infeasibility of even a home user uploading all executables for scanning periodically. Multiscan services also do not necessarily have the most optimal configuration for each antivirus product listed. This solution does not scale well to enterprise customers and often has legal limitations as well.

2.3.3 Detection.

Malware detection techniques also fit into static and dynamic analysis categories. Dynamic detection requires samples to execute in order to observe certain behaviors, while static methods rely on discovery of specific structural aspects of the program which may only be observable after a certain period of execution, such as after the unpacking process.

2.3.3.1 Static Analysis Techniques.

Academia, focused research groups, and industry have published a wide variety of static analysis techniques to detect and analyze malware. Static analysis involves looking at the executables in either packed or unpacked forms and examining the file contents or structure. File contents generally range from simple text searches to advanced techniques for finding character string matches [16] to regular expression matches [15, 16]. Files structure also includes examining program semantics, such as control flow graphs [9, 16].

Christodorescu and Jha [14] highlight difficulties commercial products have with handling simple obfuscation techniques, such as `nop` (a "no-operation" assembly instruction) insertion and inserting unconditional branches (opaque predicates as described in [19]). In their test, they apply these transformations randomly to known malware samples that three commercial antivirus products successfully detected. The commercial products tested are Norton Antivirus 7.0, McAfee VirusScan 6.01, and Command Antivirus 4.61.2. After transformation, their findings show that all three antivirus products failed to recognize ten unique mutations of the malware samples. Although they apply their obfuscation techniques to a limited sample of only four malware artifacts, the results are reasonable and likely scale.

They build a detection tool called static analyzer for executables (sic) (SAFE) [14], which takes a control flow graph as input and a malware automaton. SAFE examines the control flow graph for matches to the malware automaton, and it responds with the code sequence matching the pattern or that it found no match. They also examine how SAFE responds to normal applications based on a limited sample of four different programs. In their limited tests, they observe no false positives or negatives. Moser [65] identifies weaknesses with this method as it relies on obtaining a clean disassembly with which to construct the control flow graph.

The research of Christodorescu and Jha [14] is promising, but they cite performance as a significant weakness. In their tests, they develop the control flow graphs at load time, which takes approximately 5 seconds to build and another second to test against the small malware samples. They do not include performance data for other programs used in their tests, which imposes additional performance penalties. Analyzing much larger office productivity programs will undoubtedly take much longer. The annotation process is much larger against the 1 MB Apple *QuickTime Player* with performance reaching 800 seconds. The detector process against *QuickTime Player* took an additional 161 seconds. Limitations on the ability to automatically unpack malware samples will also decrease overall performance. Assuming average annotation and detector performances of 16 s and 2 s, a small system containing approximately $10,000$ unique executables will likely take 50 hours to scan fully. While Moore's law may have substantial impact on the original performance findings for present-day applications, this method has additional weaknesses as Moser identifies [65].

Christodorescu and Jha [15] examine other antivirus product detection rates against different types of code obfuscations. The authors apply four code obfuscations to eight Visual Basic malware samples. Although not detailed here, the results show that each antivirus product had a different false positive rate (FPR) against various obfuscation techniques. The group generally performs well against variable renaming, but much poorer against code reordering, hexadecimal encoding, and garbage insertion. The authors use correlation calculations between signature lengths and false negative rate (FNR) as key information in their algorithm.

In partnership with Carnegie Mellon, Christodorescu and Jha [16] analyze instruction semantics to detect malware. Their technique analyzes executables and uses the Ida Pro disassembler to generate the control flow graphs. Although the authors identify limited application against some obfuscation transformations, the product of this research appears robust against obfuscations. Reported performance results indicate that the detection process will take 1 to 3 minutes against relatively small malware samples, which exacerbates the overall performance problem in the scenario highlighted above.

Christodorescu, Kidd, and Goh [17] present research concerning string analysis for executables and describe a prototype program implementing their research. This research highlights the information available to the analyst by harvesting strings from binaries. In particular, strings found inside binaries at specific locations can lead analysts to discover undocumented program features and communication with the program's environment and remote systems. The working product of this research works with the Ida Pro disassembler to obtain string data references and disassembly. From the references and disassembly, their prototype considers the strings and the program manipulations of them ultimately revealing the final string values with some limitations. Many malware implementations do not contain significant string values even after unpacking. Instead, these samples "build" the final string values during runtime, which the authors' technique should detect. Simple string searches through a binary using a tool such as *streams* [74] would not find these types of strings. The authors' technique requires more overhead and specialized software, though.

Most of the Christodorescu published research [14, 16, 17] assumes that a correct disassembly of the malware samples is available, which Moser identifies as a significant weakness [65]. In the remaining research paper [15], the malware samples tested are Visual Basic applications, which an interpreter translates to machine language immediately prior to execution. This assumption is interesting considering the author's discussion of the "obfuscation-deobfuscation game" between malware authors and researchers [14]. Such assumptions cause problems when dealing with programs that keep code and data protected until needed. In such cases, code or data might remain obfuscated until needed, then re-obfuscated immediately after use [49]. The Christodorescu research uses relatively slow methods to generate the disassembly and control flow graphs, which reduce its applicability to obtaining SA in tactical situations.

Szor [91] describes many defensive strategies used in malware. He classifies many malware defensive strategies and discusses many challenges that the antivirus community faces when reverse engineering malware applications. One major defense strategy he describes is anti-emulation, where malware attempts to consume major amounts of system resources, such as nested loops, during unpacking routines. Combined with debugger detection strategies, this may cause debugger traces to simply fail or take exorbitant amounts of time to complete as resources become exhausted.

Echoing similar information, Eilam [30] describes the scenario from a general reverse engineering perspective. He describes basic and advanced software reverse engineering concepts in his book. He also details anti-disassembly and anti-debugging protections as well as malware reversing and the difficulties faced by malware defensive strategies.

Sung, Xu et al. [85, 101] use statistical techniques to analyze similarity between malware enhanced with more obfuscations and the original samples. The authors develop a prototype called Static Analyzer of Vicious Executables (SAVE) that extracts API sequence calls out of binary executables. They compare the extracted sequence to sequences of known malware. In particular, they use the Euclidian distance, sequence alignment, cosine measure, extended Jaccard measure, and Pearson's correlation measure as similarity measures. The authors claim that the underlying assumption of this research is that specific malware samples execute a "sequence of malicious API calls". An underlying assumption not stated by the authors is that the tools associated with their technique can always provide a clear disassembly. Reliance upon such an assumption causes problems given that defensive techniques described earlier are relatively simple to implement.

Xu [101] presents performance comparisons between SAVE and SAFE, which demonstrates the previous statements made about the overhead of SAFE. On the other hand, Xu claims that SAVE is 100 times more efficient than SAFE and even demonstrates analyzing Microsoft Word's primary executable *winword.exe* (unknown version), which is approximately 10 MB in size, in approximately one-half of a second.

Erdélyi [34] promotes clean booting as the best technique for detecting stealthy malware. This straightforward idea is still common today, because it denies the malware the ability to run and hide itself. The author states that clean booting an NT-based system is difficult, because the OS loads several low-level drivers even in safe mode. If one of these low-level drivers is malicious, the system will activate it possibly allowing it to evade detection. Other variants of this technique exist, such as clean booting from a bootable Linux partition or disk and then mounting the New Technology File System (NTFS) file volume for analysis.

Weber, et al. [100] develops a tool called Portable Executable Analysis Toolkit (PEAT), which helps in the analysis of malware. PEAT examines static portions of the file and provides warnings based on what it observes. For instance, it issues an alert when it sees a strange program entry point, like one in the `.reloc` section instead of the typical `.text` section. PEAT also calculates instruction frequencies and patterns, register offsets, jump and call offsets, entropy of code sections, and code or American Standard Code for Information Interchange (ASCII) code probabilities.

Moser, et al. [65] argue that a limit exists to the effectiveness of static analysis to detect malware. They introduce obfuscations called "opaque constants" which are techniques that obscure control flow and data accesses. These obfuscations prevent automated tools from generating pristine disassembly of instructions. The authors believe that dynamic analysis techniques address the shortcomings of static analysis and that static analysis alone is not reliable for classifying malware.

Bergeron, et al. [9] present a method of "static slicing" to detect malicious code. The authors claim that detection of malware requires a three-step approach of disassembling the executable, transforming the resulting disassembly to a high-level representation, and finally applying static slicing. The authors comment that they rely on the "excellent" disassemblers which are commercially available, but did not refer to the difficulty of obtaining a true, clear disassembly. They use program transformations to generate a higher-level representation, such as stack elimination and identifying function parameters. The result is an assembly-like representation, instead of a source code representation. Static slicing requires control flow and data flow graphs. Slicing results appear to capture system calls and the preceding assembly-like instruction sequences that affect call parameters.

While previously discussed research of static analysis for malware detection involves program disassembly, Treadwell and Zhou [96] explicitly avoid instruction-level semantics by focusing instead on high-level program characteristics, or anomalies. They identify seven specific anomalies in their research:

- use of non-standard section names,
- use of common packer section names,
- program entry point does not point to a section identified as code,
- non-zero Thread Local Storage virtual address in data directory,
- DLLs with no export functions,
- limited number of import functions, and
- check of import table for zero ordinal values on delayed loading APIs.

Using these anomalies, they calculate weighted total risk scores using arbitrarily assigned risk scores for each anomaly. Weights for each anomaly are inversely proportional to the anomaly frequency in non-malware samples. Their sample set consists of 2,014 non-malware and 144 malware samples. They achieve an overall detection (true positive) rate of 70.8% and a FPR of 3.872% when the total weighted risk score exceeds 1.0. Using a threshold of 4.0 for the total weighted risk score, they observe a detection rate of 29.8% with no false positives.

Rafiq and Mao [72] extract hundreds of high-level program attributes and execute a feature selection process to determine the most salient features. The following list describes some of their more prominent features:

- file size,
- presence of obfuscation,

- number of program sections,
- use of non-standard section names,
- duplicate section names,
- presence of uncommon sections,
- improper section alignments,
- Browser Helper Objects,
- running as a service, and
- including suspicious import libraries.

Using a naive-Bayes classifier with only the fifteen features selected, they exceed 90% detection accuracy with a 10% FPR. Like Treadwell and Zhou, their feature set does not necessarily include instruction-level attributes, which avoids the problems with static heuristics identified by Moser [65].

2.3.3.2 Dynamic Analysis Techniques.

Dinaburg, et al. [23] examine malware detection using hardware virtualization extensions rather than software and OS virtualization. Their research stems from a survey of obfuscation techniques of $25,000$ recent malware samples and the plausible assumption that malware can easily detect common software emulation environments. With the authors' technique, analysts can transparently and externally examine malware. They achieve transparency and externality by hosting the analysis environment in hardware virtualization, which they demonstrate in their prototype tool *Ether*. Using a Xen hypervisor, they run *Ether* at the privileged-domain level and the guest OS at user domains. The guest OSs all rely on the hypervisor for access to all hardware, but they theoretically cannot detect the hypervisor. *Ether* can monitor instructions executed, memory writes, and system calls from the guest environments. *Ether* also has the capability to limit monitoring to a specific process running in a guest OS. The authors cite architectural limitations of their hardware virtualization environment that allow possible detection based on memory flushing.

Okamoto and Ishida [69] propose a distributed agent approach to virus detection as well as virus neutralization. In their research, they harvest "self" data consisting primarily of header information and file metadata. They use autonomous, heterogeneous agents which consist of antibody (triage-type) agents, killer agents, copy agents which replace files with non-corrupt copies from elsewhere on the network, and control agents. Antibody agents compare files with their respective self data looking for compromised integrity. Killer agents simply remove files identified as corrupt by antibody agents. Copy agents replace infected, "killed" files with other non-corrupt copies from elsewhere on the network. Control agents monitor and control this advanced antivirus system. The authors test their agent antivirus architecture with five popular viruses, at least one of which appears to have been polymorphic. Their system can successfully detect all five viruses even though current antivirus products at the time cannot detect the polymorphic virus. Their agent-based system cannot neutralize against two of the viruses, because the contamination extended beyond the available self data. The authors adapt their system to include more information in the self data to deal with that strain of virus in the future. File recovery by the copy agents is successful against all five viruses tested. The authors admit that their system can require nearly double the storage space if the expansion of self data encompassed entire files.

An emerging trend in malware analysis is online automated malware sandboxes [68, 84, 95]. These services offer detailed analyses of uploaded samples. These services typically employ throttling technologies to prevent individual users from consuming too many resources. For example, Sunbelt's CWSandbox allows users to upload compressed files containing 50 samples, while Norman Sandbox only allows single file uploads. Some of these services also market stand alone products for large-scale use. The main limitations of these online sandboxes are user throttling, bandwidth consumption for large samples, analysis time required and non-definitive results. A previous section describes the functionality of these sandboxes.

These services also do not provide definitive results, except for results of antivirus scans to which they subject the sample. All of the other observation results are just raw data, which the user must interpret. Other online services exist that scan uploaded samples with numerous antivirus products, such as VirusTotal [98]. Like the online sandbox utilities, these services normally employ throttling technologies to prevent individual users from consuming too many resources. Also, these types of services do not scale much past curiosity seeker levels, because of the constraints of network bandwidth, user throttling and the sheer volume of potential samples. A major difference between online virus scans and sandbox technologies is that the former report results from antivirus vendor research, whereas the latter reports general actions and leaves the identification and classification to the user.

Individual antivirus product vendors also allow for sample uploads to their websites. These services scan files with the vendor's antivirus product. These services have the same scalability issues, but provide more definitive answers than sandboxes, because they report malware findings that the vendor has analyzed and confirmed.

2.3.4 Classification.

Antivirus researchers and companies have their own local policies and procedures for categorizing malware based on functional analyses and similarities to other known artifacts. Other third-party, nonprofit organizations propose industry standards, but vendor acceptance and adherence to the proposed standards are optional.

In general, most of the industry adopts the intent of the standards, but keep their autonomy in addressing related internal issues that the standards do not cover. Therefore, vendors commonly add new malware types to identify unknown samples if they deem it appropriate. As an example, Avira lists on its website over 60 unique malware type prefixes [6], which far exceeds the CARO standard of 10 acceptable types. Most vendors do not publish their naming convention standards, which makes their local categorization policies difficult to assess.

As previously discussed, antivirus vendors often refuse to change malware names once assigned, because it generates confusion among users. Likewise when companies change local policies, they do not recategorize previously analyzed malware, because it is infeasible given the volume of work required and suffers the same consequences. Therefore, if a particular vendor initially calls a *backdoor* sample a *Trojan*, it will not reconsider labeling the same sample if it later modified its policy to increase the precedence of *backdoors* over *Trojans*. This observation has substantial impact across any malware research effort that assumes that all historical antivirus labels are in compliance of the current vendor naming policy.

2.4 Artificial Intelligence Applications

Published research of AI applications to malware detection dates back to the early 1990s. The most common difference among the published research is the feature source. The majority use n-grams, which are byte sequences of length n normally reputed to represent instructions, but may also represent data or program structure. Although many researchers develop successful classifiers using n-grams [1, 5, 39, 46, 47, 77, 92, 94], a lack of explanation for their successes exist as some admit [46].

Other common feature sets include program instructions, API calls, anomaly data and structural information. Particularly, instruction-level information (including API calls) are elusive data features and obtaining this information can be problematic [14, 16, 65]. Execution traces take large amounts of time and other approaches make overzealous assumptions of generating a pristine disassembly to obtain the features [65].

FPRs are extremely important for malware and intrusion detection heuristic scanners [3, 5, 91], because malware and attack patterns must be observable in a deluge of normal data. Using a tool with a FPR of 0.1% with 10 real positive samples in 1 million negative samples translates to a set of 1000 negatives with only 10 true positives (assuming the detection system catches all of them). Gross alert numbers quickly overwhelm human analysts and operators or worse, condition them to not respond to the alarms. This antipathy renders such systems useless except to establish an organization's "due diligence" responsibilities.

The AI application of Kephart, et al. [44, 80] from IBM Watson Research Center to malware detection involves automated signature extraction for new variants of known malware samples. This particular research effort focuses on generating unique signatures based on specific n-grams from the executable based on the probability of finding the sequence in the malware or a non-malicious program. Although this research sought an automated solution, it is not an "intelligent" or proactive solution as it primarily focused on detecting variants of known malware.

Tesauro, et al. [94] from IBM Watson Research Center apply neural networks to successfully detect boot sector viruses. Due to significant computational and space constraints as well as a small sample size for training and validation, they use an input layer, no hidden layers and a single node output layer in their network. This approach has two major benefits, limiting the number of computations and avoiding overfitting.

To further address the runtime constraints, they restrict floating point computations to small integers. They also manipulate the decision threshold boundary to increase the cost associated with false positives as they cite that a single false positive reading likely affects thousands of systems.

They train the network with trigrams (3-byte strings) that undergo a unique feature selection process. Initially, they canvas the entire sample corpus for trigrams and eliminate all that are common to both the malicious and non-malicious sets. Moreover, they reduce the list of trigram features to the set where each malicious training sample contains at least four trigrams. This selection process leads to a three order of magnitude feature reduction.

Expanding on their previous work, Arnold and Tesauro [5] incorporate a voting system on multiple trained neural networks. By training multiple networks with distinct features not used in others (termed *boosting*), they effectively avoid the major pitfall associated with heuristic scanners, high FPRs. Their assumption is that these disparately trained networks rarely produce identical false positives. Szor [91] cites that the Arnold and Tesauro network research has such a low FPR that Symantec incorporated it into its antivirus product default scanning.

Luke and Harris [52] investigate using intelligent agents to detect unknown malware. The authors claim that mobile intelligent agents are "less vulnerable to attack" and have greater visibility and redundancy. Their research considers detection of viruses during replication and determining similarity between "code sentences," presumably a byte representation of a sequence of instructions. They assumed that the distribution of agents in the network was uniform and that the agents detect viruses by observing differences between the same types of file. The authors state that the feature space to examine exhibits a problem in dimensionality, which is why the authors chose to use Albus' Cerebellar Model Articulation Controller (CMAC) algorithm [2], a descendant of the perceptron commonly used in artificial neural networks. CMAC is a lattice Associative Memory Network. They highlight advantages of CMAC versus standard neural networks, but they report no significant research achievements thus far.

Using three different feature sources to identify malware, Schultz, et al. [77] test different data mining algorithms against standard signatures. In their first approach, they examine information from the portable executable (PE) header as features, such as import libraries and the number of imported functions from those libraries, with Cohen's improved rule learning algorithm called RIPPER [18]. This method requires unpacking the samples before evaluation to reveal the true imports, but the authors do not refer to this step. The second approach uses strings found in the binaries as features, which is again problematic without first unpacking. The third method captures byte sequences presumably expected to translate loosely to a representation of instructions. The authors used both the string and byte sequence data with naive and multi-naive Bayes classifiers. The results of the naive Bayes classifier with the string features is the most accurate classifier in their tests reaching a detection rate of 97.43% with a FPR of 3.80%.

Finally, Schultz, et al. [77] concede that encryption (and presumably packing) obfuscate strings present in the executables, but the solution they suggest makes bold assumptions. They indicate that an effective method of handling the packing case is to initially assume that a sample is malicious and then if strings are found in the program the classifier defaults back to the naive Bayes algorithm. One point the authors fail to mention is that the attacker has complete access to the malware sample. The attacker can use any encryption method to make the ciphertext code look however they want (e.g., one-time pad ciphers) or "stuff" the unpacking area with byte sequences of their choosing, such as normal instructions from a normal program.

The unpacking stub overwrites the unpacking area anyway making this almost trivial. Another bold assumption the authors make is that a sample is malicious if strings are not present (i.e., a packed executable), but many legitimate programs are also packed and contain no significant strings.

Abou-Assaleh, et al. [1] uses Common N-Gram (CNG) analysis to classify malware. The authors examine character n-grams found in executables in order to capture features associated with the author and tools used to write or compile the code. The authors make their observations on a small sample size of only 25 and 40 samples of malicious and benign code respectively. With the prevalence of malware publicly available, one might question why the authors do not consider a larger sample making their findings more significant. The authors claim that the character n-gram technique produced 100 percent accuracy on training data and 98 percent accuracy on test data, but the test data performance appears to be a best-case scenario.

Kolter and Maloof [46] examine the results of several classifiers on malware detection via n-grams. Techniques they test include naive Bayes, support vector machine (SVM), DTs and boosted variants of each. In their experiments, they evaluate the classifier performance by computing the area under a receiver operating characteristic (ROC) curve. Their boosted DT model achieved the best accuracy, an area under the ROC curve of 0.996. The authors cite that identifying why the presence of some byte strings combined with the absence of other byte strings contributes to high performance classifiers is difficult.

Research in behavioral classification for Microsoft by Lee and Mody [49] encourages runtime behavior-based automated classification. According to the authors, "Automatic malware classification is becoming an important research area" [49]. They state its importance is due to the incredible growth of new malware variants. Lee and Mody suggest that optimal classifications should come from using results from both static and dynamic analysis. They state that “the complexity in extracting features out of malware is a difficult task,” which has the agreement of others [41, 65, 72].

Using “opaque objects”, they develop semantic representations of objects. They calculate similarity distances between objects for use in cluster analysis. They use nearest neighbor classification and compute an adapted Levenshtein Distance [50] measure between known cluster representatives and new exemplar data. The authors identify several limitations of their research including clustering time and space complexity and inadequacies of the similarity measure chosen to include semantics and inter-process relationships. They cite a best performance error rate of eight percent [49].

In order to demonstrate intra-family support without falling under an inter-family support threshold, Henchiri and Japkowicz [39] propose an exhaustive search for n-gram features. This automated feature selection process provides a limited set of features from the training sets in k-fold cross validation. They use both ID3 and J48 (a C4.5 variant) DT algorithms among other techniques. (Waikato Environment for Knowledge Analysis (WEKA) [37] calls their DT implementation J48.) They compare their classifier results with [77] for the same sample corpus and achieve an advantage in overall accuracy improvements of up to 29.77% in comparable tests. Not only do the DT algorithms achieve the greatest improvements, but they also have the greatest overall accuracies and the lowest FPRs. In tests with minimal sequence length and least inter-family support, ID3 achieves a 99.77% detection accuracy with a 0.34% error rate.

Bailey, et al. [7] investigate the disparities between malware family identification across antivirus products. They confirm the same difficulties identified in previous sections about the problems with naming conventions. The authors execute malware samples in a virtual machine and record their behavior, but they focus on the lasting state changes rather than the OS system calls as much of the other dynamic analysis work does. Finally, a cluster analysis identifies similarities between samples and forms the basis for an appropriate label. Labeling consistency leads to improved communication concerning malware samples according to Bailey.

The Intelligent Malware Detection System (IMDS) is the association rules classifier of Ye, et al. [102] that they compare to the classification performance of Naive Bayes, SVM and DT classifiers. They specifically examine the top 500 API calls of maximum relevance, using their presence or absence as features. The classifier accuracy rate of their IMDS prototype is 0.9307. The authors reveal that their implementation is now part of KingSoft Antivirus Software.

Rafiq and Mao [72] implement Naive Bayes classifiers with high level features to detect malware. Their features include file size, obfuscation indicators, the number and names of program sections, and section alignment all of which come from the PE header with the possible sole exception of obfuscation indicators. The authors claim that if a sample program is a Browser Helper Object (BHO) in their study, then their system classifies it as malware with a 98% likelihood. A BHO is a browser plug-in for Microsoft Internet Explorer to extend its functionality. Other features the authors use are whether the executable runs as a service and if it imports specific DLLs.

They implement a feature extraction system with access to hundreds of features. Their Naive Bayes classifier uses different combinations of features from a sample set of approximately $7,000$ programs with k-fold cross validation. With a set of samples where half are malicious, they report a mean experimental detection rate of approximately 90%.

A recent patent application from Schipka [76] on behalf of MessageLabs Ltd. shows a litany of classification functions they use to identify malware in various types of message streams, including e-mail, hypertext transfer protocol (HTTP), file transfer protocol (FTP), and others. Specific techniques cited include linear classifiers, Bayesian, neural networks, SVMs, k-nearest neighbors, radial basis function neural networks, and genetic algorithms and evolutionary systems. Potential features include literal values from fields, such as header fields, as well as computed values.

Tabish, et al. [92] focus on distinguishing malware from specific types of files. They test non-malicious documents, executables, pictures, movie files, portable documents and compressed files to a malware set of six different types. The classes of malware they assess include backdoors, Trojans, viruses, worms, constructors and miscellaneous malware. They divide samples into blocks and compute a variety of statistics to use as features for a boosted J48 classifier for each block. After correlating block predictions, their system makes malware predictions based on the percentage of malicious blocks. Their system achieves a 90% detection accuracy for models distinguishing each malware class from the assortment of non-malicious files. The malware classes chosen by Tabish, et al. are a mixture of malware propagation methods and payloads.

In the sole malware static heuristic research application to determine malware payloads, Kolter and Maloof [47] extend their previous detection work [46] to identifying three specific types of malware. They identify the difficulties associated with manually acquiring appropriate labeling for the malware types from analysis reports, which limits their test to 525 samples of backdoors, mass-mailers and viruses. To prevent undersampling by using compound class labels such as *backdoor+mass-mailer*, they use one-versus-all classification, where a sample assumes a positive label if it has a specific capability, such as backdoor capability or mass-mailer. If the backdoor and mass-mailer classifiers predict positives, the overall prediction becomes *backdoor+mass-mailer*. Using stratified sampling and ten-fold cross validation, they report mean area under curve (AUC) and 95% confidence intervals of 0.8986 ± 0.0145 using a

SVM classifier, 0.8704 ± 0.0161 using a boosted J48 classifier and 0.9114 ± 0.0166 using boosted J48 as best performing classifiers for mass-mailers, backdoors and viruses respectively. Interestingly, there is no statistically significant difference between SVM and boosted J48 performance in their tests. The malware classes chosen by Kolter and Maloof are a mixture of malware propagation methods and payloads.

2.5 Situation Awareness

Endsley [33] is a renowned expert on situation awareness (SA) and her research dominates the literature. Due to the ubiquity of the Endsley SA model, this dissertation only examines MaTR implications for this model although the mappings and discussion likely apply equally to other SA models as well.

Endsley defines SA as "the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future" [33]. She explicitly separates *situation assessment* as the "process of achieving, acquiring, or maintaining SA" [33]. SA is only relevant in dynamic environments and does not include static knowledge representations, such as procedures and checklists.

Endsley describes the human property of attention as a limitation for obtaining SA. She asserts that humans have a limited bandwidth for direct attention, which is necessary at all SA levels. Complex or stressful situations may easily consume a decision-maker's attention bandwidth and have adverse implications on the operator achieving or maintaining SA.

Figure 9 shows Endsley's SA model [31, 33]. Abilities, experience and training are factors external to SA that provide a foundation to mental processes that ultimately influences decision making. These factors distinguish between novice and expert operators. Experienced decision-makers can make poor decisions without accurate and

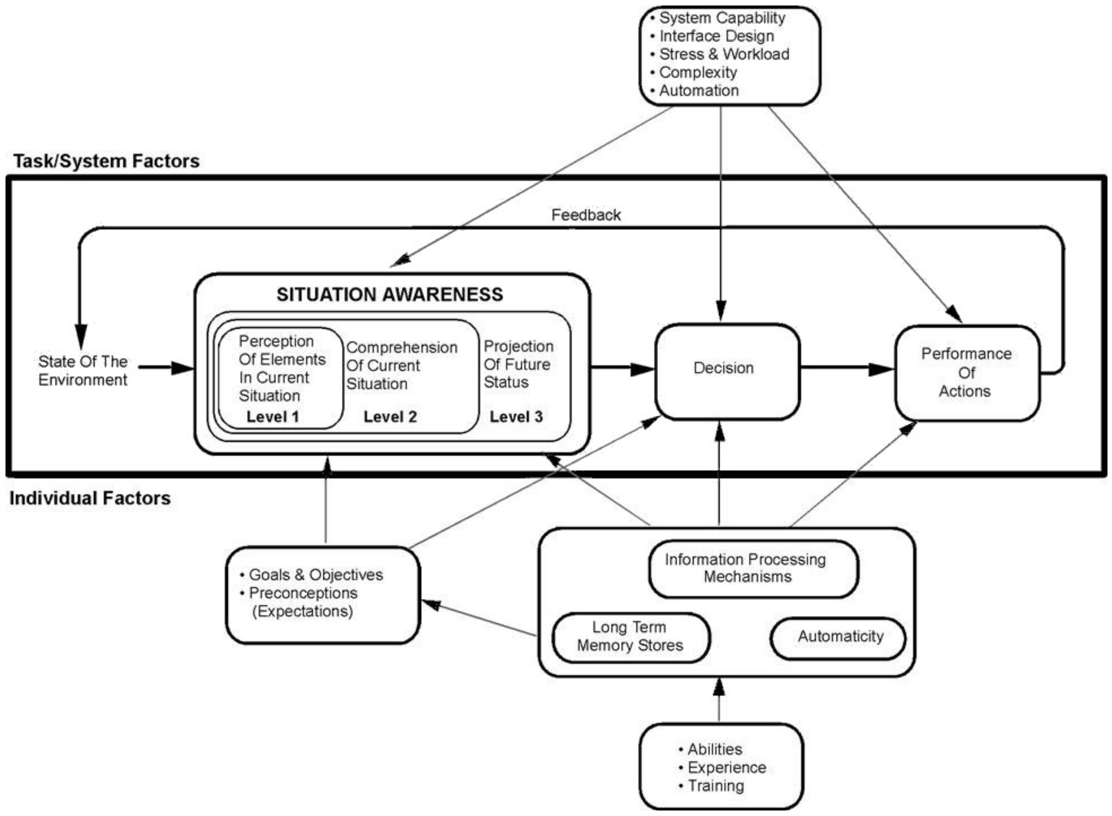

Figure 9. Endsley's SA model (adapted from [33]).

complete SA, but novices can make poor decisions with perfect SA. In the latter case, the novice needs additional experience or training in order to make good decisions.

Abilities, experience and training contribute to the forming of schemata and mental models in long term memory [33]. These schemata and mental models are generic patterns of types of situations that one uses to identify appropriate responses based on the current goal. Endsley refers to these responses as *plans.* If operators have previously experienced a situation, they may have stored *scripts* in long term memory which are lists of actions to achieve end states compatible with current goals. When scripts are available to execute a plan via experience or training, the operator can execute the script to accomplish his goal. The Endsley SA model from Figure 9 represents schemata, mental models, plans and scripts as "Preconceptions".

Goals are self-explanatory, but particular goals may focus operator attention to specific elements while lowering attentiveness to others, which is a phenomenon Endsley refers to as automaticity. Automaticity [33] is a mental "auto-pilot" mode, such as when one drives home from work but cannot remember specific situation details like stopping at red lights, etc. The potential danger of functioning in automaticity is a reduced sensitivity (due to lack of attention) to new stimuli in the environment.

Perception (Level 1). The first level of the Endsley SA model is perception, the knowledge of the state of significant elements in the environment. This state information includes both static and dynamic attributes. An example of an element at the Level 1 SA is a defender observing Michael Jordan at half court dribbling a basketball while in full stride with a determined expression on his face (and a grin).

Comprehension (Level 2). Comprehension is the fusion of Level 1 information to generate the proverbial "big picture" in the operator's mind. Novices may have the same Level 1 SA abilities as veteran operators, but may be incapable of assimilating the information to form a Level 2 SA. Comprehension is the interpretation of what the current environment state *means* and is the logical justification for potential response actions. Continuing from the previous example, Michael Jordan still has the ball, his team is down by one point, only five seconds remain on the game clock and the observer is the only defender on the defensive end of the court. All of these disparate Level 1 elements now clearly describe a tactical scenario.

Projection (Level 3). To achieve what Endsley refers to as Level 3 SA, one needs the ability to project the future state of the environment especially in the immediate future, as tactical situations often require. The grand purpose of reaching Level 3 SA is that it provides the operator the opportunity and knowledge to determine the best course of action to meet their objectives [33]. A likely future projection from the previous example is that Michael Jordan is going to embarrass the poor defender as he scores the game winning basket with a spectacular slam dunk as the game ends. This projection allows the defender to determine his best course of action, no matter how unappealing those options might be.

Endsley also identifies task and system factors that influence SA and the decision making process [33]. System capability is a measurement of how well the system enables SA by obtaining state information about pertinent elements in the environment. The MaTR architecture provides the operator information directly related to its major goals, versus antivirus products providing only a vendor-assigned name. Common human stress factors may cause the operator to limit attention to a small set of elements at the expense of others, which may be non-beneficial to SA if the attended elements are not the most salient for achieving SA in the current environment. Full automation may lead to lower SA during automated solution failure or degradation. Focused automated solutions that diminish the operator's effort with workload and SA acquisition are believed to be beneficial for achieving SA.

Interface design recommendations from Endsley demonstrate the process of obtaining SA, or situation assessment. The primary objective of the system interface is to provide the operator with information necessary to achieve SA without exhausting their attention bandwidth [33]. Following is a list of interface designs most pertinent to this research that leads to systems that have positive effects on operator SA hypothetically:

- degree of interface capability to provide information contributing to Level 2 or Level 3 without unnecessary operator effort
- "degree to which information is presented in terms of operator's major goals" [33]

- make critical cues for mental model and schemata activation salient
- always provide a global SA view across goals and detailed information supporting immediate goals
- system projection support for novice operators
- parallel processing design for sharing attention

One particular design factor Endsley describes is complexity [33]. As systems become increasingly more complex, the additional components, communications, and dynamics of the components require greater amounts of operator mental workload to achieve the SA levels. As cyberspace is an extremely dynamic environment, the rate of communications, complexity of end systems and activity certainly exceeds the ability of humans to perceive, comprehend and make projections of the environment without targeted automation to enhance human effectiveness.

2.6 Confusion Matrix Statistics

A confusion matrix is a common method to record classification performance. This research uses the following confusion matrix statistics for comparison between methods and to quantify performance. Table 5 is a sample confusion matrix with predictions specified by columns and actual classes indicated by row entries.

For confusion matrices where $n = 2$ classes and where Class 2 is considered "positive", Equations 25-28 are the FPR, true positive rate, FNR and true negative rate, respectively. For $n = 2$, $entry_{1,1}$ and $entry_{2,2}$ represent the number of true negatives and true positives respectively, while $entry_{1,2}$ and $entry_{2,1}$ represent the number of false positives and false negatives respectively.

Table 5. Sample confusion matrix for n-class classifier results.

		Predictions		
	Class	Class 1	$\ldots$	Class n
Actual	Class 1	$entry_{1,1}$	$\ldots$	$entry_{1,n}$
	$\ldots$	$\ldots$	$\ldots$	$\ldots$
	Class n	$entry_{n,1}$	$\ldots$	$entry_{n,n}$

$$\text{false positive rate} = \frac{entry_{1,2}}{entry_{1,1} + entry_{1,2}} \tag{25}$$

$$\text{true positive rate} = \frac{entry_{2,2}}{entry_{2,1} + entry_{2,2}} \tag{26}$$

$$\text{false negative rate} = \frac{entry_{2,1}}{entry_{2,1} + entry_{2,2}} \tag{27}$$

$$\text{true negative rate} = \frac{entry_{1,1}}{entry_{1,1} + entry_{1,2}} \tag{28}$$

For multi-class problems where $n > 2$ classes, the above equations are inadequate to express performance except for all-but-one classifiers as no single, "positive" class exists. Instead of positive and negative rates, the producer's and consumer's accuracies similarly describe the results, except they generalize to a larger number of classes. Equations 29-32 are the producer's accuracy, omission error, consumer's accuracy and commission error for Class c, respectively. The producer's accuracy and omission error have a truth perspective, while consumer's accuracy and commission error have a user perspective. The producer's accuracy and omission error for a "negative" class are synonymous with true negative rate and FPR respectively in the two-class problem. For the "positive" class in a two-class problem, the producer's accuracy and omission error are synonymous with true positive rate and FNR respectively.

$$\text{producer's accuracy} = \frac{entry_{c,c}}{\displaystyle\sum_{m=1}^{n} entry_{c,m}} \tag{29}$$

$$\text{omission error} = \frac{\displaystyle\sum_{m=1}^{n} entry_{c,m} - entry_{c,c}}{\displaystyle\sum_{m=1}^{n} entry_{c,m}} \tag{30}$$

$$\text{consumer's accuracy} = \frac{entry_{c,c}}{\displaystyle\sum_{m=1}^{n} entry_{m,c}} \tag{31}$$

$$\text{commission error} = \frac{\displaystyle\sum_{m=1}^{n} entry_{m,c} - entry_{c,c}}{\displaystyle\sum_{m=1}^{n} entry_{m,c}} \tag{32}$$

The kappa statistic [81] is a measure of agreement between two judgments. Equation 33 is the formula for computing the kappa (κ) statistic, where $P(agree)$ is the probability of agreement and $P(chance)$ is the probability of chance agreement between the two judgments, such as truth data and classifier predictions. The range of the kappa statistic is $[-1, 1]$, where 1 indicates perfect agreement between predicted and actual classes and -1 is perfect disagreement. When $\kappa = 0$, no correlation between predicted and actual classes exists.

$$\kappa = \frac{P(agree) - P(chance)}{1 - P(chance)} \tag{33}$$

2.7 Summary

This chapter describes in depth many of the popular machine learning techniques found in malware detection research. Several machine learning techniques exist that show promise in the malware detection problem. Many of the machine learning techniques have application to both continuous and discrete datasets. As signature-based solutions become overwhelmed by an exponential growth in malware, pattern recognition based solutions are gaining popularity.

The majority of published research in malware detection with machine learning techniques involve the use of n-grams, short sequences of bytes commonly found in each class of software. Although many researchers develop successful classifiers using n-grams, a lack of explanation for their successes exist as some admit [46]. Many other techniques examine elusive data features, such as instruction-level information or subroutines, but obtaining this information is problematic [14, 16]. Execution traces take too much time and other assumptions about pristine disassembly make overzealous assumptions [65].

III. Methodology

3.1 Chapter Overview

This chapter explains how this research accomplishes its goal and experimentally defines the MaTR architecture for further evaluation. Development of the cyberspace machine learning classifiers as targeted sensors for SA follows the CRoss-Industry Standard Process for Data Mining (CRISP-DM) [13]. This process ties the development of classifiers back to the underlying business objectives.

This chapter first briefly introduces the fundamental process steps of the CRISP-DM. After describing process execution for this effort, the chapter concludes with a thorough discussion of the resulting classifier models.

3.2 CRISP-DM Process

The CRISP-DM process is a six-phase representation of a data mining lifecycle commonly used in industry [13]. Figure 10 is an adaptation from the CRISP-DM consortium [13] illustrating the iterative nature of this process. The outer circle with arrows describes data mining in general and how it is often an iterative process with subsequent iterations capitalizing on previous knowledge gleaned.

One can intuit common activities of each CRISP-DM process phase by the specific phase name. The following paragraphs present brief descriptions of each phase and their typical activities.

Business Understanding. The CRISP-DM process [13] begins with defining the high-level business objectives and the criteria for success to ensure the analyst knows what the organization actually wants to achieve. This step avoids wasting time

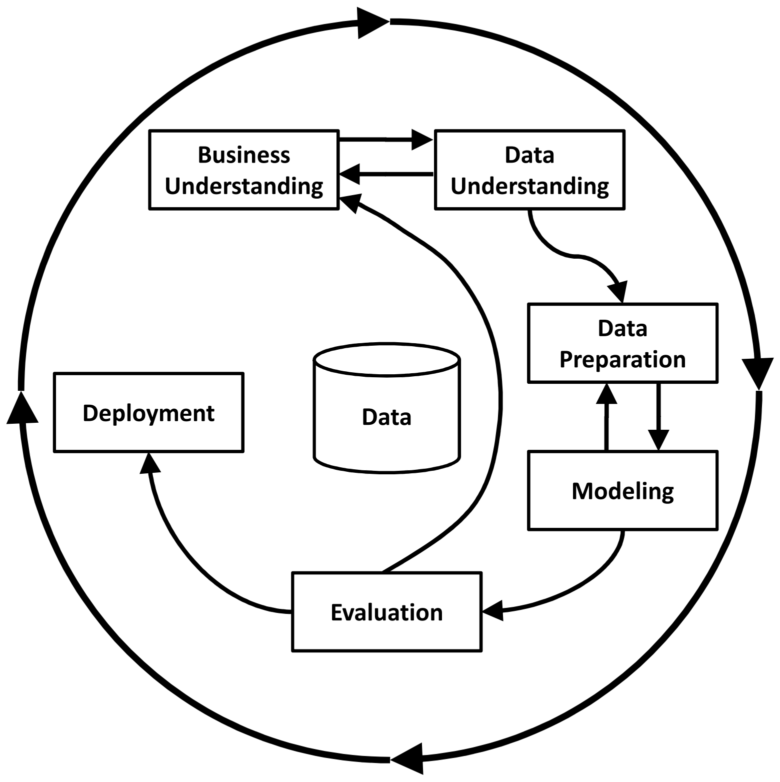

Figure 10. Adaptation of CRISP-DM process (from [13]).

and resources while not pursuing the purpose of the activity. Part of the knowledge discovery may include answering key questions regarding business operations.

The end result of the business understanding phase is the analyst gaining a thorough low-level understanding of the business problem. The analyst documents available resources, problem constraints and assumptions. Collectively this information frames the problem space and enables the analyst to define specific data mining goals and success criteria to meet the business objectives. One key output of this phase is a project plan that is a roadmap for successfully accomplishing business goals by meeting the data mining goals.

Data Understanding. The data understanding phase involves the acquisition, exploration and quality control of the dataset itself. The analyst defines the collection method and describes the data representation and units of measure. If necessary, the analyst considers integration steps to deal with combining data from multiple data sources. The next task involves exploration of the dataset, such as determining distributions and relationships among features, which may lead to the disqualification of specific models for the problem. The final task is to verify the data's quality, including completeness, presence of errors and non-duplication of exemplars among other subtasks. The distinction between portions of the data understanding phase and the data preparation phase may blur depending on the application.

Data Preparation. Determining which features and exemplars to include from the dataset is the initial task of the data preparation phase. Selected data may undergo a cleaning process, such as inserting estimates for missing data. The next task is to construct the data, including computing derived features, projection transformations, data binning and generating synthetic data. Integrating and merging data tasks may be necessary as previously alluded and may involve generating longer, more expressive tuples by matching primary and foreign keys of records of multiple tables in a relational database. The final task of the data preparation phase is formatting the data allowing for proper import to the modeling tool.

Modeling. The modeling phase requires the analyst to select appropriate models suited for the problem based on objectives and whether characteristics of the data agree with assumptions of the candidate model. The analyst should define how they will test model performance before building the model. Dataset divisions may include validation sets in addition to the standard training and test sets. Generating the model includes selecting appropriate model parameters. After building the model, the analyst assesses whether or not the model meets the established success criteria.

Evaluation. Evaluation of the model determines if it achieves the stated business objectives. The decision to reject the model may involve previously unperceived business constraints, which the analyst may incorporate into the next data mining iteration. A review process is necessary to ensure model development adhered to the business and model requirements. Evaluation ends with a determination of the best course of action, whether it is model deployment or additional iterations are justified.

Deployment. This phase defines the deployment strategy based on evaluation. An additional plan published during deployment is the monitoring and maintenance plan, which describes how to monitor the deployed model for incorrect usage or ineffectiveness over time. The last task of the deployment phase is the delivery of the final report and lessons learned.

The six phases are themselves iterative, because phase transitions depend largely on the results of the previous phase. For example, a specific data mining goal is to produce a classifier to identify faulty widgets with 95% accuracy during modeling. If the model accuracy on test data is only 89% in the evaluation phase, the discrepancy may warrant further business or data analysis or possibly deployment because the last several iterations yielded no substantial improvement.

3.3 Execution of the CRISP-DM Process

This section describes the execution of the CRISP-DM process until the generation of the final model architecture. Sections with the appropriate phase heading include discoveries and observations made about appropriate architectural decisions during the particular phase regardless of iteration. The scope of this dissertation does not require the deployment phase and the evaluation phase is limited to the discussion of how the final MaTR model architecture contributes to cyberspace SA according to the Endsley model [33].

3.3.1 Business Understanding.

Chapter I introduces the motivation and the background scenario for this multi-faceted research problem. Observations of the current CND scenario imply many significant requirements for the development of cyberspace SA indicators. This section describes these observations and the model requirements derived from them.

A major enabling requirement for cyberspace SA is the ability to detect malware. The final MaTR architecture must be capable of detecting malware with a high level of accuracy. Therefore, model accuracy is a major statistic of observation. Taken further, the FPR and FNR are also critically important to cyberspace SA. The FPR directly relates to the wasteful overhead on CND operators and defensive analysts. Given the large number of unique, non-malicious programs running on organization networks, an acceptable FPR realistically depends on the available operator or analyst capacity and the number of unique, non-malicious programs. The FNR relates to missed malware samples, which diminishes cyberspace SA and allows threat operations to continue unnoticed.

Another problem with current capabilities is the dearth of effective tactical indicators. Commercial antivirus products, one of the major resources currently available to address this problem, are effective against known threats, but they are "slow" scanners, because they have to compare static indicators to identification and behavior signature databases [87]. Dynamic analysis methods requiring CPU emulation are even slower [87]. These products address this by restricting searches to the most probable infection points [87, 91] unless the user opts for a full scan, which may take hours to complete and is not suited to time-sensitive tactical scenarios. As indicators for cyberspace SA, the MaTR architecture must have excellent runtime performance to be valuable in tactical situations. Tactical situations, such as air-to-air combat, have relatively short durations and tactical situations in the cyberspace domain are no exception. As a result, the MaTR architecture restricts itself to only static heuristic features that do not require runtime observation or CPU emulation [91]. MaTR also restricts feature sources from instruction-based static heuristics which are difficult to acquire [65].

If the MaTR architecture can communicate rationale for its predictions clearly, it becomes more trusted and usable by the human operator. Rather than attempt to model dozens of low-level contextual elements, which would have a negative impact on runtime performance, MaTR further restricts itself to a feature set based on commonly used practitioner information (anomalies and structure), which simplifies human understanding of its decision making process. This restriction enhances MaTR's human-machine communication potential, because it only uses features that experienced operators already understand. One commonly used feature set in the literature that does not follow this principle is n-grams. Kolter and Maloof allude to the difficulty of understanding how n-gram classifiers make predictions by following a decision path including or excluding particular byte sequences [46].

Antivirus products routinely only provide simple alerts, such as present or not present. Even using their heuristic scanners, antivirus products routinely only report whether or not a sample's attributes exceeds a preset threshold. Knowing how close any given sample is to exceeding the threshold adds substantial context and allows the operator to make a judgment call particularly in scenarios requiring higher sensitivities. The final MaTR model should provide additional information to provide operator flexibility and allow additional data fusion methods.

3.3.2 Data Understanding and Data Preparation.

The dataset source is a collection of 32-bit PE samples obtained from well known sources. All “clean” samples come from harvesting of clean installs from vendor media of Microsoft Windows XP, Vista, and Windows 7, while the malware samples come from an updated download of the VX Heavens dataset [99]. Specifically, the malware, or “dirty”, samples are Trojans, worms, and viruses as identified by the antivirus label assigned to them. Extractions of PEs from these sources eventually yields 25,195 clean and 31,193 dirty samples for a total of 56,388 samples available for testing of the final MaTR model. Earlier pilot tests relied on relatively smaller samplings, because the PE collection was still in its infancy. During the last process iteration, the final models had much larger samplings available for testing.

This project uses a custom program to extract high-level program features from the PE collection. This program also computes derived features, such as anomaly and certain structural attributes. Direct comparison of extracted data with output of other tools [40] and code [12] verifies the feature values are correct. The resulting dataset is not multivariate normal per the Jarque-Bera [58] and Lilliefors [59] goodness-of-fit normality tests, because nearly all features are not normal [66]. This observation may cause problems for machine learning techniques that assume normality, such as LDFs, but should not impact others.

Verifying data correctness after integration is an important issue for ensuring the validity of experimental results. Extraction of features from the non-malware and malware sets uses the same tool to simplify data integration. Additional measures to assure proper data integration is the extensive use of hashes to avoid including duplicate exemplars.

Determining appropriate exemplar labels is another significant challenge. While most antivirus products agree on general malware determinations [25], they often disagree on more specific identifications, such as malware types. An initial pilot investigation of the level of malware type agreement between three antivirus products shows no practically significant difference in FNN classifier performance with labels determined by partial and full agreement. The study finds a statistically significant difference of only 1% accuracy between partial and full agreement labeling. This result implies that different antivirus vendor labels are more appropriate for identifying additional interpretations rather than determining a single label to assign. Section 4.4 describes the labeling methodology used in the MaTR achitecture for determining propagation methods and payloads.

This project uses MATLAB to develop classifier models. As MATLAB operates primarily on double-precision floating point numbers, the final number formats for dataset features are doubles. Again, the feature extraction tool handles this conversion.

3.3.3 Modeling.

The modeling phase is extensive and attempts to answer key questions for project success. Determining the proper architectural approach to problem decomposition has a potentially huge impact on the success of this research, because building accurate classifiers is paramount to increasing SA. Different architectural approaches to the problem may yield better classifiers and have a more definitive impact on cyberspace SA. This effort experimentally determines important architecture decisions, such as the top performing classification algorithm and its parameters allowing focus to shift to establishing an engineering advantage for the MaTR architecture and its enhancement of cyberspace SA.

These pilot tests use lower K values of three or five for cross validation to allow for sufficient representation of smaller class populations in test sets. Test designs involving the final model all use ten-fold cross validation. Each test iteration begins with a random selection of ten equally-sized folds. For each K-fold run, one of the ten folds serves as the test set while the remaining folds comprise the training set.

3.3.3.1 Modeling Approach and Best Classifier Model.

Two major architectural approaches exist to provide additional malware type context. The first approach is a "parallel" method that uses a single classifier to distinguish non-maliciousness files from specific malware types (direct typing). The "series" approach first performs the detection task followed by a sequence of specific type checks. This test only examines the detection task and excludes the sequence of specific type checks. The former approach may have better runtime performance as a single classifier, whereas the latter is potentially more accurate as it can focus exclusively on distinguishing between malware classes without interference from the volume of non-malware samples. Table 6 shows the class populations for these pilot tests.

Table 6. Sample summary for multiple classifier test.

Class	Number of Samples
Non-malware (NM)	11,714
Backdoor (BD)	3,359
Trojan (T)	930
Virus (V)	973
Total	16,976

The pilot test also evaluates four different machine learning classifiers, specifically LDF, DDA, FNN, and DT. These experiments include DTs, because they demonstrate superior performance in other research [1, 39, 46, 47]. This test examines LDF due to observed singularity despite the appropriateness to use QDF when class covariance matrices are different [28]. The LDF performs better due to the QDF singularity issues as Tadjudin and Landgrebe suggest [93].

The DDA experiments rely heavily on MATLAB's histogram function for feature binning. In order to minimize the cell sparseness problem [82], this process restricts the number of total bin enumerations to less than the smallest number of observations for a class. Due to the combinatorial explosion of possible values for each attribute and its potential to cause cell sparseness, the DDA model only uses three of the more salient features as determined by other pilot tests and previous experiments.

Following Duda, et al. [28] recommendations, the FNN structure for the detection and direct typing problems includes a single hidden layer of 56 and 52 nodes respectively to meet network weight to sample ratios. The network also uses hyperbolic tangent activation functions for input and hidden layers and pure linear activation functions for the output layer. The input layer consists of 21 nodes and the output layer for the detection problem is 1 node and 4 nodes (one for each class from Table 6) for the direct typing problem. The network uses gradient descent with momentum during learning with a learn rate of 0.1 and momentum constant of 0.9. All loss functions are mean squared error in spite of the recommendation from Kivinen and Warmuth to use relative entropy [45], because of implementation difficulties in MATLAB and time constraints. By using the mean squared error as the loss function, the resulting FNN models may exhibit suboptimal performance. As determined empirically, training lasts for a maximum of 100 epochs as determined by pilot tests with a performance goal of 0.005.

The classification DT model uses the standard MATLAB implementation with default parameters, including pruning. The impurity function computes Gini's diversity index [28] to determine split variables. The minimum number of samples associated with a node to allow the node to split is ten samples. Pilot tests confirm these model parameters produce a model with better performance compared to other parameter combinations tested.

The experiment separately tests each classification method using five-fold cross validation to minimize sample bias. Furthermore, five replications with new fold selections reduce experiment variance, which leads to narrower confidence intervals for analysis based on preliminary tests. Specific folds also use stratified random sampling to ensure representative samples for each class in the test.

In general, the experimental results are favorable for all of the classifiers compared to static heuristic malware detection results from Chapter II, but the DT models clearly perform best. The FNN classifier performs comparably, but has some distinct disadvantages, notably a much higher training overhead and making zero predictions for the Trojan class. The disparity between the results from the detection and direct typing problems across all applied techniques implies that an initial detection test followed by a series of one-versus-all classifications is the best approach to providing additional threat information. Distinguishing between non-malware and malware is noticeably easier and appears to dominate the distinctions between the more similar malware types during training. All of the techniques demonstrate this same result, which supports this theory.

Table 7 recaps the performance results of each technique for the different problems. The DT superior performance is statistically and practically significant for both applications, but the FNN performs comparably. For each respective problem, the performance differences between models tested are all statistically significant. Both LDF and DDA performed similarly overall, but LDF outperformed DDA substantially with the non-backdoor malware types.

All resulting confusion matrices for these tests are in the Appendix along with the confusion matrix statistics for the detection problem. Tables 8, 9, 10 and 11 are the resulting confusion matrix statistics for each classification method on the direct typing problem. All of the techniques show difficulties differentiating the Trojan

Table 7. Accuracy results summary with confidence intervals.

Method	Problem	Mean Accuracy	95% Confidence Interval
LDF	Detection	0.8746	0.8697 — 0.8792
DDA	Detection	0.8659	0.8642 — 0.8675
FNN	Detection	0.9451	0.9418 — 0.9483
DT	Detection	0.9764	0.9755 — 0.9774
LDF	Direct Typing	0.7776	0.7750 — 0.7802
DDA	Direct Typing	0.7844	0.7823 — 0.7864
FNN	Direct Typing	0.8799	0.8769 — 0.8830
DT	Direct Typing	0.8984	0.8962 — 0.9006

class from the others. In these experiments, LDF and DT are the most accurate, demonstrating still lackluster producer accuracy rates of 0.5049 and 0.4439 for the Trojan class respectively. As discussed in previous experiments [25], the disparities among researchers and antivirus vendors for the Trojan class may contribute to this performance degradation.

3.3.3.2 Model Parameter Assessment.

In this pilot test, MaTR uses a logical, two-phased architectural approach of first identifying whether an executable sample is malicious and then determining its most likely type. Malware detection is a straightforward two-class problem with distinct malware types grouped together under a single malware class, *M*. The malware typing problem uses the distinct malware types provided by antivirus scans. The dataset is a concatenation of static feature extractions from 32-bit non-malicious executables and malware of the following types: backdoors, downloaders, Trojans, password stealers, worms, droppers and viruses.

The MaTR classifier model is the MATLAB implementation of the classification DT based on its flexibility when dealing with data from non-normal distributions and categorical data as well as its performance in previous experiments. The purposes of the following experiments are to verify classifier detection accuracy using MaTR's set

Table 8. LDF test confusion matrix statistics for direct typing problem.

Statistic	NM	BD	T	V
Producer Accuracy	0.9032	0.5112	0.5049	0.4452
Consumer Accuracy	0.9513	0.7680	0.1707	0.4994
Omission Error	0.0968	0.4888	0.4951	0.5548
Commission Error	0.0487	0.2320	0.8293	0.5006
Kappa	0.5639			

Table 9. DDA test confusion matrix statistics for direct typing problem.

Statistic	NM	BD	T	V
Producer Accuracy	0.9241	0.7416	0	0.0002
Consumer Accuracy	0.8864	0.5230	0	0.3333
Omission Error	0.0759	0.2584	1.0000	0.9998
Commission Error	0.1136	0.4770	1.0000	0.6667
Kappa	0.5188			

Table 10. FNN test confusion matrix statistics for direct typing problem.

Statistic	NM	BD	T	V
Producer Accuracy	0.9729	0.8741	0.0013	0.6206
Consumer Accuracy	0.9522	0.7053	0.2000	0.7207
Omission Error	0.0271	0.1259	0.9987	0.3794
Commission Error	0.0478	0.2947	0.8000	0.2793
Kappa	0.7402			

Table 11. DT test confusion matrix statistics for direct typing problem.

Statistic	NM	BD	T	V
Producer Accuracy	0.9784	0.7977	0.4439	0.7176
Consumer Accuracy	0.9836	0.8033	0.3886	0.7542
Omission Error	0.0216	0.2023	0.5561	0.2824
Commission Error	0.0164	0.1967	0.6114	0.2458
Kappa	0.7888			

of static heuristic features and to extend the application to malware typing. Both experiments are 3^2 factorial designs testing the effect of two significant factors in decision trees, minimum parent split and split criterion values. The three treatment levels chosen for the minimum parent split value are 10, 20 and 30 based on data from limited preliminary tests. The three treatment levels chosen for split criterion value correspond to the available functions in MATLAB [57]: Gini's diversity index, the twoing rule and the maximum deviance reduction.

The minimum parent split value defines the cutoff threshold value for determining if the number of samples associated with a node in the tree warrants possible splitting. Lower cutoff values commonly lead to overfitting and loss of generalization [28]. On the other hand, larger minimum parent split values may prevent the tree from identifying significant patterns in the data or underfitting. The split criterion value maps to a MATLAB function the tree employs to measure impurity and determine the splitting feature and value.

Both experiments also use five-fold cross validation with fifty replications. Each replication includes a new random sampling to determine folds for subsequent cross validation runs. To avoid overinflation of the detection results, sampling adheres to a fixed 2:1 $NM{:}M$ ratio to accommodate DT model requirements. The malware samples are stratified random samplings from the malware types tested.

Pilot Study - Detection Classifier Results. The results for the detection classifier are quite significant as all models exceed a 0.99 apparent test accuracy rate with equally impressive FPR and FNR. The resulting mean confusion matrix for this test is in the Appendix. Table 12 shows the confusion matrix statistics for the trained model with the best treatment combination. The mean apparent test accuracy rate for this model over the cross validation and replication runs is 0.9935 with a kappa

Table 12. **Detection confusion matrix statistics.**

Class	Producer Accuracy	Consumer Accuracy	Omission Error	Commission Error
NM	0.9949	0.9955	0.0051	0.0045
M	0.9909	0.9898	0.0091	0.0102

statistic of 0.9855, which demonstrates "almost perfect" agreement between model prediction and actual classes based on the characterization of Landis and Koch [48].

The producer accuracy of 0.9949 for class *NM* is a high enough specificity to avoid deluging analysts with as many false positives as similar published research [72, 96]. While no standard, acceptable FPR threshold exists for this application, Anderson [3] echoes the need for minimizing the FPR for intrusion detection system applications. Furthermore, the producer accuracy of 0.9909 for class *M* is a high enough sensitivity to limit overlooking potential malware. Although this experiment uses a uniform cost of misclassification, a cost matrix adjustment can shift the FPR and FNR depending on operational needs.

The model parameter test examines the performance impact of two factors, minimum parent split value (A) and split criterion (B). According to the analysis of variance (ANOVA) results in Table 13, strong evidence leads to rejection of H_0 concerning equality of the main effects for factors A and B, but no evidence suggests a significant interaction effect between these factors. All ANOVA assumptions are met with residuals fitting a normal distribution and having constant variance.

Figure 11 shows the mean comparison for different treatment level combinations with confidence intervals. Expectedly, the classifier performance improves as the minimum number of samples required for splitting decreases for the treatment levels tested. This trend is readily apparent in the mean comparison plot as the three lowest accuracies have the highest minimum parent split values (Factor A). All treatment

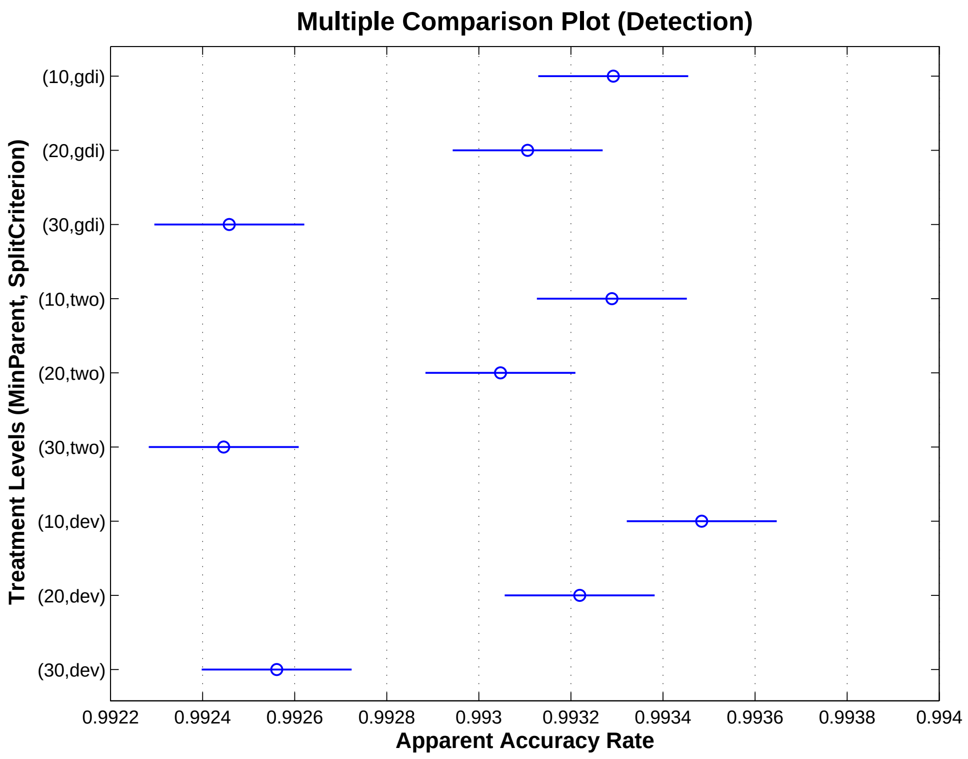

Figure 11. Comparison of detection parameters.

levels with cutoff values of 30 have reduced performance that is statistically significant from all other treatment combinations.

The treatment level combinations with values for Factor A of 20 exhibit a decreased performance from points with values for Factor A of 10, but not enough statistical evidence exists to make a definitive claim of which is better. Both of these sets of combinations likely occur nearer to the "knee" of a performance curve than

Table 13. ANOVA results for detection model.

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	3.02e-04	2	1.51e-04	88.4	1.09e-37
B	1.12e-05	2	5.61e-06	3.29	0.0376
A*B	8.30e-07	4	2.07e-07	0.121	0.975
Error	3.83e-03	2241	1.71e-06		
Total	4.14e-03	2249			

the minimum parent split values of 30, which may explain the significant difference between these data points and not the others.

The only remaining statistically significant treatment combination is $(10, dev)$, where the parameters are the values for factors A and B respectively. This treatment level combination yields the best result and is significantly different from the $(20, gdi)$ and $(20, two)$ combinations. This data point fits with another observable trend, the general improvement in performance for Factor B levels from *two* to *gdi* to *dev*.

These detection results show that MaTR demonstrates results similar to the other malware detection research using static analysis techniques and features. The next section describes the extension MaTR makes to the malware typing problem.

Pilot Study - Typing Classifier Results. While seemingly not as spectacular as the detection tests, the malware typing model provides modest performance relative to the previous detection results. Considering the predictions do not require any lengthy manual inspection process and only use static heuristics, the typing results show a strong potential for prioritization of analysis backlogs. The resulting mean confusion matrix for this 7-class problem is in the Appendix. Table 14 shows the confusion matrix statistics for the model with the best treatment level combination. The mean apparent test accuracy rate for this model over the cross validation and replication runs is 0.5904 with a kappa statistic of 0.4738, which demonstrates "moderate" agreement between model prediction and actual classes based on the characterization of Landis and Koch [48]. Although not extremely high, the expected value of a random classifier is only 0.1429 given seven malware classes tested.

Although the typing model fails to achieve high accuracy on the specific malware types, it still demonstrates potential value for identifying backdoors (BD), downloaders (DW) and virus (V) classes. Classifier performance for identifying true Trojan

Table 14. Typing confusion matrix statistics.

Class	Producer Accuracy	Consumer Accuracy	Omission Error	Commission Error
Backdoor (BD)	0.7510	0.7033	0.2490	0.2967
Downloader (DW)	0.6593	0.6366	0.3407	0.3634
Trojan (TJ)	0.3678	0.3589	0.6322	0.6411
Password Stealer (PS)	0.4553	0.4993	0.5447	0.5007
Worm (W)	0.4283	0.4548	0.5717	0.5452
Dropper (DR)	0.3599	0.4282	0.6401	0.5718
Virus (V)	0.5630	0.6742	0.4370	0.3258

(TJ) and dropper (DR) samples is especially poor with producer accuracies of 0.3678 and 0.3599 respectively.

Antivirus applications often classify the same samples inconsistently, which can negatively affect the results of this typing experiment, because the supervised learning relies on the type labels from antivirus scans. Future investigation may examine the performance impact of confounding malware types together and vendor labeling disparities to regain high confidence in the results and maximize situation awareness. For instance, classifier inconsistencies between the *backdoor* and *Trojan* classes exhibit the highest error concentration and account for 5% of all typing errors.

Another explanation for the difficulty the model has classifying between malware types is a possibly inherent inadequacy in the salient value of the static heuristic feature set for making such determinations. The feature set may contain enough informational value to provide a high degree of accuracy for detection, but that may be its limit.

The model parameter test examines the performance impact of two factors, minimum parent split value (A) and split criterion (B). According to the ANOVA results in Table 15, strong evidence leads to rejection of H_0 concerning equality of the main

Table 15. ANOVA results for typing model.

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	7.65e-02	2	3.83e-02	891	2.15e-285
B	2.49e-03	2	1.20e-03	28.9	3.89e-13
A*B	4.98e-05	4	1.24e-05	0.290	0.885
Error	9.62e-02	2241	4.29e-05		
Total	1.75e-01	2249			

effects for A and B, but no evidence suggests a significant interaction effect between these factors. All ANOVA assumptions for this test are met with similar residual normal and variance plots from the detection test.

Figure 12 shows the mean comparison for different treatment level combinations with confidence intervals. In this case, strong evidence suggests the treatment combination that exhibits the best performance mean for treatment combination $(10, gdi)$ is different. No significant difference exists between any of the *two* and *dev* levels for factor B with the same level for factor A (e.g., $(30, two)$ and $(30, dev)$).

This plot shows general patterns similar to the detection results. One observable trend is the general improvement in performance from levels *two* to *dev* to *gdi* for factor B, only slightly different than the detection results. Another general improvement trend is across treatment levels for factor A with lower split values exhibiting significant performance improvements.

3.3.3.3 Model Assessment Methods.

When feasible, re-accomplishing other research tests generates competing models and establishes a baseline performance standard for the final MaTR model. These tests are truly apples-to-apples comparisons of the two models using exactly the same K-folds and metric collection techniques. If resources prohibit this approach, statistical comparison with the published results is the only method possible.

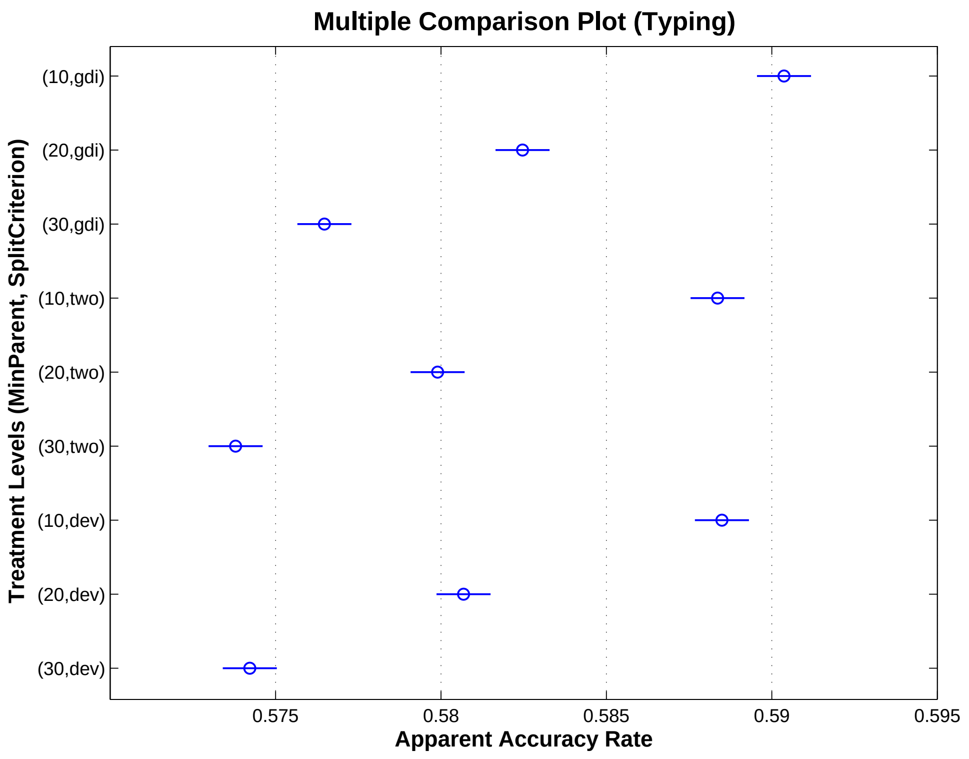

Figure 12. Comparison of typing parameters.

3.3.4 Evaluation.

The evaluation phase describes how Endsley's SA model [33] relates to cyberspace SA and the implications of the final model's performance on that mapping. Chapter IV includes a discussion of the mapping after presenting the results of the final model's performance. Finally, a scenario illustrates its potential impact.

3.4 Final Model Architecture

The final MaTR model architecture uses a straightforward process for detecting malware using only a program's high-level anomaly and structural data. While many researchers and commercial companies use this same structural data, none rely exclusively on this source of data and achieve the performance levels of MaTR. Figure 13 shows the inputs and outputs of MaTR and illustrates its internal process. Inputs to MaTR are executable files, such as PE files common in the Microsoft Windows OS.

Although an open system, MaTR explicitly bounds the machine and human operator together within the overall system, a subtle yet significant distinction from other work that simply uses a computer to generate "answers". In MaTR's architecture, the operator becomes a critical component receiving and providing feedback to the rest of the system and eventually initiating a response action. Limiting features to historically relevant information is a requirement to maximize potential feedback with a human operator. One can visualize this benefit when considering the comprehension difficulty for a human faced with the resulting decision process of an n-gram solution or the lack of decision making information provided by a commercial antivirus product that only provides the final result. The co-alignment of the human operator and the machine within MaTR allows for critical and constructive feedback, which remains an area for continued work.

The "Data Pre-processing" stage allows for any steps required before feature extraction and subsequent classifications. Data pre-processing actions include discovery of valid executable files. Other actions include pre-filtering known malware and known non-malware, decrypting data, and data sharing with other sensor systems.

During "Feature Extraction", the system parses the input file to find the predetermined data inputs for the subsequent classifiers. Features (described later) are restricted to the input file's high-level structural anomalies and raw structure information. "Feature Transformation" involves any action taken on features before classification, such as bounding, mapping, projecting, etc. Examples of well known transformations include principal component analysis and factor analysis.

The "Detection Classifier Data" component represents the data for the trained classifier. For example, DT classifiers must correctly initialize a binary tree node

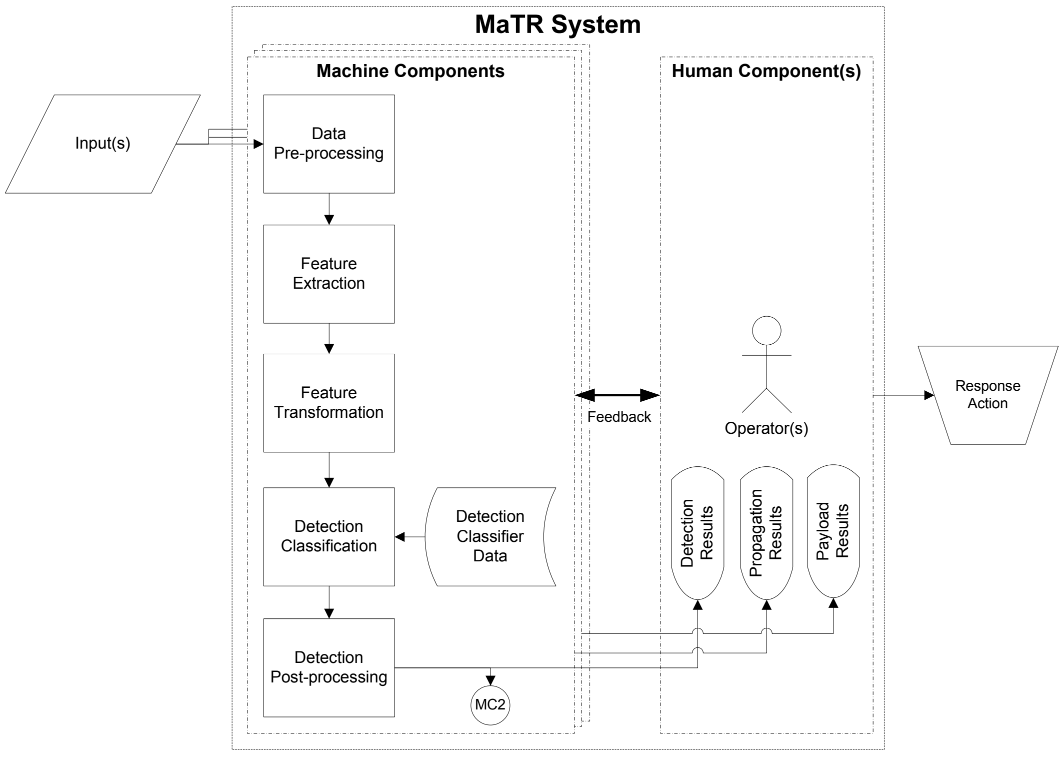

Figure 13. MaTR system process.

structure with appropriate cut features to uniquely identify the specific feature to test, cut values and classification decisions for the leaf nodes of the DT.

The underlying DT classifier comprises the "Detection Classification" component. At this point, the classifier takes the transformed features and makes its classification decision based on its underlying algorithm. For example, in DTs, the decision sequence begins at the root node and progresses down to a single leaf node where the classification decision is determined. "Detection Post-processing" allows for post-filtering before presenting preliminary results to the operator, triggering additional actions, result verification with other systems, or data fusion with additional sensors.

As MaTR does not rely on computations to determine the final feature set, it avoids the overhead of a resource-intensive feature selection step [46]. However, the Kolter and Maloof method results in a simpler and more efficient feature extraction step. This results in a trade-off as MaTR's feature extraction process remains more complex throughout its life cycle.

3.4.1 Feature Overview.

Perhaps the greatest distinction between MaTR and other research products is its feature source. MaTR achieves high detection performance while restricting its features exclusively to high-level program structural anomalies and general structural data. Instead of following a mathematical model to determine features, MaTR utilizes features commonly used by analysts [72, 91, 96] when examining samples to determine if they are indeed malicious. Rafiq and Mao found that malware routinely contains structural anomalies (78%), while non-malware does not (5%) [72].

The term "high-level" structural data refers to the basic structural format that the OS loader uses when loading an executable program into memory before runtime and higher level information, such as common file attributes (e.g., name, path, file size, etc.). The sources for the structural anomalies come from a number of publications and observations of program structure. Combining expert experience with program structural information capitalizes on analysts experience while allowing for identification of additional anomalous feature combinations.

As analysts examine samples, their previous experiences contribute to a prior knowledge of analysis technique effectiveness and past observations. Significant observations useful for confirming malice are anomalies primarily seen in malware. Routinely, analysts combine available anomaly information with structural information to either confirm their suspicion or look for additional anomalies. For instance, if the visible program disassembly is insufficient to provide any significant advertised function, the analyst may suspect that the program is packed. Many popular packers dedicate a program section for unpacking, but the section must allow reading and executing (as it will soon contain code), but it must also allow writing to unpack the obfuscated code before attempting to execute it. Analysts confirm these section permissions, or characteristics, by examining structural information for yet another anomaly.

MaTR utilizes over 100 static heuristic features based on structural anomalies and structural information itself. Many of MaTR's features are interval, unlike the Kolter and Maloof method which uses exclusively Boolean features. MaTR does not attempt to generate an instruction disassembly due to the difficulty of validating its correctness [65] nor does MaTR use instruction sequence signatures as commercial antivirus programs use [87].

Structural anomalies are generally logical operations on program header information or file areas pointed to by header information. Classes of structural anomalies include: section names [72, 91, 96], section characteristics [72, 91, 96], entry point [91, 96], imports [72, 77, 96], exports [96], and alignment [91]. Structure information, included to enable classifiers to identify additional anomalous combinations, comes directly from the PE headers, such as the `IMAGE_FILE_HEADER` and the `IMAGE_OPTIONAL_HEADER` [62]. A description of some of the more popular anomaly features follows.

Non-standard section names. Several researchers [72, 91, 96] also identify the presence of a non-standard section name as anomalous. Microsoft [62] defines several standard section names for PEs and most compilers adopt this standard. This standardization has led to an overwhelming majority of non-malware containing only standard section names. According to Rafiq and Mao [72], only 3% of non-malware use unconventional section names, while 80% of malware samples use non-standard names.

Non-standard section characteristics. Several researchers [72, 91, 96] identify non-standard section characteristics as an anomaly. If a code section has read, execute and *write* characteristics instead of the normal read and execute characteristics, it immediately raises analysts' suspicions. Normally, the program uses sections with these permissions to unpack obfuscated code before attempting to execute it. This particular anomaly is common in malware, because packing is a common malware armoring technique [91].

Entry points. A program entry point that points to a section not marked as containing code is anomalous [96]. Szor states that a program whose entry point does not point to the code section (`.text` for default compiling) is another entry point anomaly [91]. Packers commonly adjust the entry point to point to an additional code section to start the unpacking process.

Imports. Inclusion of information regarding import libraries and functions is common among malware research [72, 91, 96]. Common features include the numbers of import libraries and functions. Executables with a low number of imported functions are suspicious [96], because programmers normally provide program utility by importing functions, such as I/O, encryption or complex math.

Exports. Treadwell and Zhou also identify DLLs that export no functions as anomalous [96]. Since the purpose of a dynamically-linked library is to provide functionality to other programs via exported functions, the absence of exported functions is surely suspicious.

3.5 Chapter Summary

This chapter explains how this research addresses the problem area by using a standard industry-defined process and then assessing how the resulting model can positively impact cyberspace SA. The CRISP-DM process serves as a guide for data mining projects by defining typical tasks and suggesting methods to remedy shortcomings for each project phase. A summary of experimental findings during iterations of data mining follows the process description. The final model is one possible result of the process and serves as the test model for experiments described in Chapter IV. While the CRISP-DM process normally ends with deployment of a solution, this dissertation ends in the evaluation phase with the identification of how the final MaTR prototype addresses the business problem of limited SA given its performance characteristics.

IV. Results and Analysis

4.1 Overview

This chapter presents experimental results of the final model of the MaTR architecture from Chapter III. Experimental results show engineering advantages of MaTR over commercial solutions and other research in malware static heuristics in terms of accuracy performance, execution time and training time. Finally, the chapter covers the mapping of the problem area to the Endsley SA model and discusses the model's potential impact on enhancing cyberspace situation awareness (SA).

Comparisons between MaTR and the Kolter and Maloof [46, 47] n-gram methodology are appropriate, because both methods exclusively use static heuristic features to make predictions and the published results of Kolter and Maloof are the most definitive. Kolter and Maloof are also the only other researchers who expressly build and test classifiers capable of identifying specific malware types or functionality.

Both the Kolter and Maloof research and MaTR appear best suited for providing strong indicators of the presence of malware to achieve cyberspace SA. Deployments of both methods are extremely efficient making them practical for split second, on-demand feedback for any given sample. On the other hand, dynamic analysis methods and slower static analysis methods that must generate disassembly or control flow graphs may take on the order of seconds to make predictions for a single sample.

The purpose of these experiments is to demonstrate engineering advantages of the final MaTR model from Chapter III. The first experiment examines the performance advantage of MaTR versus the Kolter and Maloof [46, 47] n-gram detection methodology. For better comparison, this effort reproduces the original Kolter and Maloof work with a much larger dataset (original work uses 1,971 benign and 1,651 malware samples; retest uses 25,195 benign and 31,193 malware samples) and employs additional experiment standardization techniques. For example, using the exact same folds for cross validation reduces experimental variance and use of identical statistic collection methods for both models ensures consistency when collecting metrics.

The next experiment examines MaTR's detection accuracy against a validation set of relatively unknown malware versus the n-gram methodology and three major commercial antivirus products. The test includes performance metrics from three different sensitivity levels for all detection methods. Finally, this experiment presents results of combining the three commercial products into a "super" antivirus product and examines its performance against MaTR with various sensitivities.

The last experiment is an evaluation of MaTR's performance for identifying malware propagation methods and payloads. An initial examination compares MaTR's performance against original work from Kolter and Maloof [47] in predicting payloads via n-grams. The MaTR experiment formally focuses its attention on the propagation methods that the antivirus industry has informally adopted [64, 86, 91]. Next, another MaTR extension examines performance against a set of strategic payloads.

The chapter concludes with a thorough description of how MaTR can assist operators in achieving cyberspace SA in regards to malware. The standard for relating MaTR performance to SA is the ubiquitous Endsley SA model [33].

4.2 Comparison to n-gram Detection Methodology

The following experiment is a thorough performance comparison of MaTR and a retest of the Kolter and Maloof n-gram detection research [46, 47]. The retest uses a much larger sampling than their original work and validates their claim that performance results should improve with additional samples. The results demonstrate a significant performance improvement for MaTR in terms of operational utility.

4.2.1 Methodology.

Section 3.4 already describes the MaTR methodology. This section covers the Kolter and Maloof [46, 47] methodology for generating n-grams, a commonly used data source for text classification.

In reconstructing the Kolter and Maloof experiment, this research uses their described methodology [46] to generate n-grams and employs their identified length of $n = 4$ with a 1-byte sliding window. They treat the presence of an indicated n-gram as a Boolean feature to their boosted DT classifier. Tests utilize only the 500 most relevant n-grams based on information gains as computed by the following formula:

$$IG(j) = \sum_{v_j \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_j, C_i) \log \frac{P(v_j, C_i)}{P(v_j)P(C_i)}, \tag{34}$$

where C_i is the ith class and v_j indicates the presence or absence of the jth n-gram. The prior and conditional probabilities are self-explanatory.

4.2.2 Data Collection and Features.

The following experiments examine only 32-bit PE samples obtained from well known sources. All "clean" samples come from harvesting of clean installs and updates of Microsoft Windows XP, Vista, and Windows 7, while the malware samples come from the VX Heavens dataset [99] in April 2010. The malware, or "dirty", samples include Trojan, worm, and virus types as identified by the antivirus label assigned to them. Extractions of all 32-bit PEs from these sources yields 25,195 clean and 31,193 dirty samples for a total of 56,388 samples. These tests do not currently use samples from SourceForge as in [46], because of perceived potential for including malware in the clean sample corpus based on [71] versus harvesting exclusively from vendor media and updates.

Kolter and Maloof use `hexdump` to capture n-grams in their experiments [46, 47], but `hexdump` has documented side effects when dumping hexadecimal representations when using format strings to control output [54]. Specifically, when the format string calls for 4 bytes and the sample has less than 4 bytes remaining, `hexdump` zero-pads the output. To avoid this error, a heavily optimized, custom program (developed to perform this function) extracts the n-grams for this experiment given the requirements from Kolter and Maloof [46, 47]. This program satisfies their requirements by determining class prior probabilities and extracting all 4-grams present in each PE with a 1-byte sliding window. The program correctly handles the case where `hexdump` potentially fails ($fileSize \texttt{ mod } 4 \neq 0$) by not padding the output to create *phantom* n-grams. Finally, the program computes the information gain of each 4-gram based on Equation 34 and lists the 500 highest gain 4-grams.

4.2.3 Experimental Design.

This experiment is a side-by-side comparison of leading static analysis malware detection techniques, specifically MaTR and the previous Kolter and Maloof n-gram research [46, 47]. For consistency with prior research, these tests both adopt a standard experimental design using stratified, ten-fold cross validation. Each disjoint fold contains roughly the same number of samples from malware and non-malware sets. During each run, a different fold functions as the test set while the remaining folds comprise the training set.

Each fold requires a determination of the top 500 n-grams specific to that fold's training set for the Kolter and Maloof technique. Classifiers train on only the samples from a fold's training set and test results come from application of the trained classifier to the fold's test set. MaTR and the Kolter and Maloof retest use identical folds.

These experiments use DTs since both Kolter and Maloof [46] and Dube et al. [26] both identify these techniques as the best performing for this problem set in previous work. Both techniques use the ensemble `TreeBagger` classifier from MATLAB [60] with 25 trees. These tests use only `TreeBagger` default parameters of a minimum of one observation per leaf and $\sqrt{n}$ features selected at random for each cut variable, where n is the total number of features. The major differences in the Kolter and Maloof retest and the original work [46] are the use of MATLAB's `TreeBagger` instead of the boosted `J48` DT implementation in WEKA [37] and the larger sampling. The Kolter and Maloof original work [46, 47] uses 1,971 benign and 1,651 malware samples, while the retest uses 25,195 benign and 31,193 malware samples.

4.2.4 Measures of Effectiveness.

In order to fully compare these experimental results to other published research, this effort includes the results of a variety of measures commonly used in this research area. Kolter and Maloof report receiver operating characteristic (ROC) area under curve (AUC) results with confidence intervals [46]. Schultz et al. report accuracy with FPR, but also provide confusion matrix data, which allows for calculating FNR [77]. Tesauro et al. include general values of accuracy and FPR [94]. ROC measures use pooled averages across folds as described by Maloof [53]. Computed confidence intervals result from studentized bootstrapping with the sampling defined by the ten cross-validation folds.

Comparison of performance data from the Kolter and Maloof retest with the original work serves as a validation measure for the experiment. Given the high accuracy Kolter and Maloof achieve in their tests (AUC of 0.9958), any significant lesser performance in the retest would require a full ANOVA using the `J48` solution in WEKA

[37] and substantially smaller sample sizes to determine the the source of variation. Kolter and Maloof claim that larger sample sizes would increase performance [46].

4.2.5 Other Observations.

This section describes non-performance observations from this experiment. Kolter and Maloof [46, 47] allude to these in their original work. In particular, this section describes the number of unique n-grams generated with the larger sampling, a brief discussion of the n-grams with the highest information gain and final model characteristics.

4.2.5.1 Data Collection Observations.

Using the described Kolter and Maloof parameters for generating n-grams yields a mean of 2,103,005,388 unique n-grams across training sets. Given that $n = 4$ bytes, the maximum possible number of unique n-grams is $2^{32} = 4,294,967,296$. The observed number of unique n-grams utilizes (or saturates) 49% of the possible 4-gram space. For a perspective of retest scale, Kolter and Maloof observe a saturation rate of only 6% in their large dataset for their original experiment.

Determining the set of n-grams using the Kolter and Maloof method requires extensive "computational overhead" as they attest [46]. The datasets become too large to store in memory and as a result, calculations must resort to heavy disk-utilization with deliberate runtime performance optimization. The number of expected unique n-grams is a critical implementation factor, as it is key in determining how best to partition the n-gram data space.

In this experiment, the Kolter and Maloof n-gram generation technique generates a mean of 2,103,005,388 unique n-grams across training sets. This results in a larger saturation level of $2,103,005,388/2^{32} = 49\%$ compared to the saturation level of

6% from the Kolter and Maloof research [46]. While this saturation level causes complications for determining the top n-grams to select, it does not impede the Kolter and Maloof model classification performance, because the saturation of the n-gram space does not affect final model decisions which occur in the leaves of the DTs. Theoretically, their model uses 500 Boolean features which yields $2^{500} = 3.27e150$ potential leaf combinations given the DT classifier.

Figure 14 is a plot showing the number of unique n-grams growing as the number of files parsed increases. Unfortunately, the two have a clearly linear relationship for the range tested with a strong Pearson's correlation of 0.9950. The larger sample sizes forces calculations to predominantly disk-based solutions.

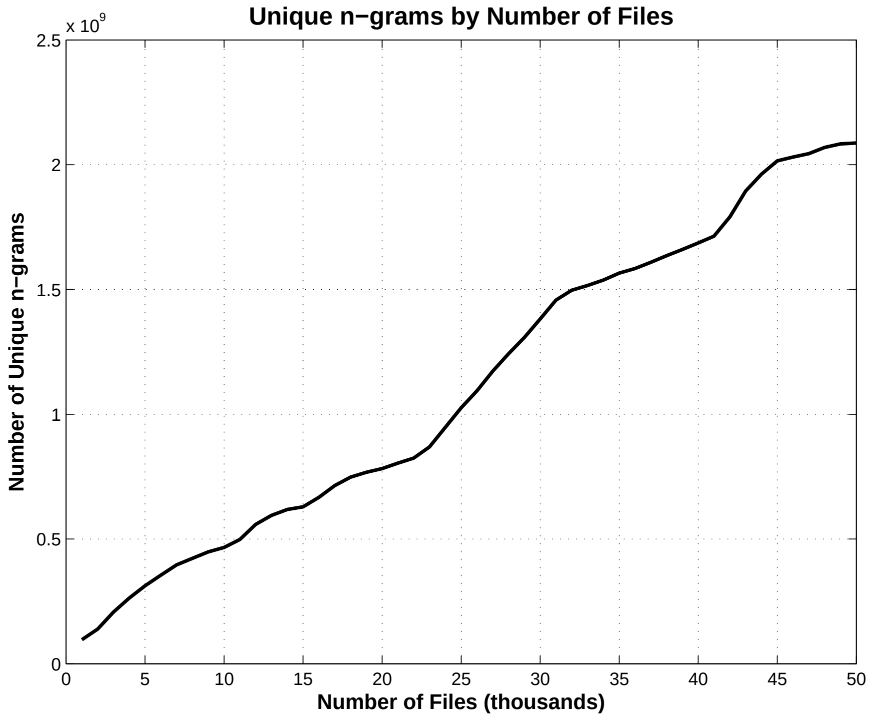

Figure 14. Number of unique n-grams increases linearly.

4.2.5.2 Selected *n*-gram Observations.

The Kolter and Maloof method selects a total of only 505 unique n-grams to use as features across all ten folds making fold selections quite consistent. Table 16 shows the top seven n-grams for all folds. The primary difference of the remaining n-grams across folds is their order.

Table 16. Top seven *n*-grams across all folds.

0x00560001
0x56000100
0x72007900
0x00720079
0x0043006c
0x43006c00
0x44006500

One observation about this partial listing is that the selected n-grams appear to focus on capturing specific byte sequences peculiar to each class. For instance, the first n-gram `0x00560001` is a 1-byte shift from the second n-gram chosen `0x56000100`. This pattern propagates through the selections with potentially longer byte sequences daisy-chained together.

A second observation is the prevalence of zero bytes (`0x00`) throughout the selections. Nearly 44% of all selected n-gram bytes are zero bytes. Closer examination of the zero bytes reveals a potential pattern of Unicode character representations, zero bytes followed by non-zero bytes. This pattern is visible in 79% of all n-grams selected.

Kolter and Maloof describe the difficulty in validating why n-grams work for classifying PEs [46]. As an example, they found a mysterious n-gram (`0x0000000a`) in their studies [46, 47], which they can not attribute as being code, data, or structure. This specific n-gram `0x0000000a` is found in a comparable percentage of samples in the expanded malware set from VX Heavens as Kolter and Maloof found [46, 47] (75% in their work), but the same n-gram also appears in 83% of the non-malware set and the information gain feature selection algorithm never ranks it in the top 500 for any folds. Why Kolter and Maloof focus on this particular n-gram is uncertain as they do not specify whether their calculations lead to its inclusion in the top 500 n-grams.

MaTR avoids the validation problem by using only historically useful static heuristic information [72, 91, 96] as features as described in Section 3.4.1. Using common anomalies and irrefutable structural information that analysts routinely use in making their assessments provides strong validation of MaTR's results. As a result, an analyst can confirm its decisions based on meaningful observations. The major difficulty with interpreting MaTR's decisions is the complexity of following the logical steps of an ensemble method—even the apparently intuitive DTs.

4.2.5.3 Model Observations.

The resulting classifiers from the original Kolter and Maloof research are ensembles of small trees [46], averaging 90 nodes. In the ensemble of DTs in Kolter and Maloof retest, the tree sizes are much larger averaging 2,824 nodes per tree. Given the 49% saturation of the 4-gram space and the much larger sampling in the retest, the trees likely had to grow substantially to minimize impurity at the leaf nodes.

MaTR averages 354 nodes per tree for the trained ensemble of DTs in these tests, which is approximately 3 times the tree size observed in previous MaTR research with smaller datasets [26]. The simpler tree representations of MaTR are likely due to the expressive power of augmenting the Boolean features with interval features. For instance, Boolean features are incapable of representing distances without additional Boolean features with special encoding. As an example, to represent the number of import libraries for a PE may require multiple Boolean features to establish data bins, such as $\{[0..5], (5..200], (200..\infty)\}$. The trees in the Kolter and Maloof retest have inefficient representations as all features are Boolean, which forces trees to grow significantly larger to accommodate the increased saturation of the n-gram space.

4.2.6 Experimental Results and Discussion.

Figure 15 shows a magnification of the ROC curves for both MaTR and the Kolter and Maloof n-gram retest. While both methods demonstrate excellent results, MaTR achieves the more ideal ROC curve as it tracks closer to the left and top sides, resulting in a mean AUC of 0.999914 for MaTR compared to 0.999173 for the Kolter and Maloof retest. Furthermore, MaTR never exhibits a lower true positive rate (TPR) or a higher FPR than the Kolter and Maloof retest for any of the 2,500 threshold values tested for the ROC plot. While the resulting AUC performance difference is statistically significant, it is not necessarily practically significant as both methods are close to ideal.

Tables 17 and 18 are the resulting AUC and accuracy confidence intervals for MaTR, the Kolter and Maloof retest, and past research. The AUC results for the Kolter and Maloof retest are a statistically significant 0.34% mean difference from their original research [46, 47]. This observation is quite interesting considering the increased saturation of the possible n-gram space for this larger test, but the classifier adequately compensates by extending the length of branches to utilize more of the available combination space.

Table 17. Mean AUC and confidence intervals.

Method	**Mean**	**95% CI**
MaTR	0.999914	0.999840 — 0.999987
KM retest	0.999173	0.998926 — 0.999421
KM original [46, 47]	0.9958	0.9934 — 0.9982

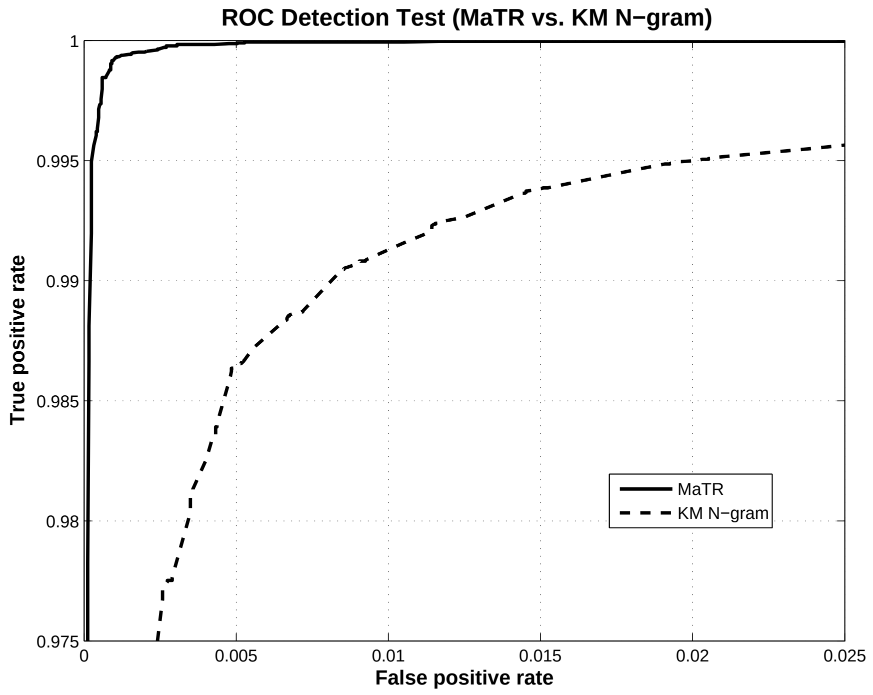

Figure 15. ROC curves for MaTR and KM *n*-gram retest.

Table 18. Mean accuracy and confidence intervals.

Method	**Mean**	**95% CI**
MaTR	0.999166	0.999007 — 0.999325
KM retest	0.989919	0.988897 — 0.990941
Schultz (strings) [77]	0.9711	*not reported*
Schultz (DLL function calls) [77]	0.8936	*not reported*

Although the confidence intervals for MaTR and the Kolter and Maloof retest are close, MaTR demonstrates superior results that are statistically significant to both the Kolter and Maloof original and the retest. This consistency may indicate a higher saliency value of structural and anomaly data for detecting malware than *n*-grams, which are typically used for text classification. However, both results strongly suggest that static heuristic methods remain viable for malware detection.

For comparison with other research, Table 18 includes the apparent accuracy statistics. MaTR's accuracy is significantly better than those for the Kolter and Maloof retest as the confidence intervals do not overlap. While MaTR's accuracy is consistent with its AUC results, the Kolter and Maloof retest reveals an unexplainably lower accuracy than one may anticipate. Analysis of the additional metric now leads to practically significant results as the accuracy results of the Kolter and Maloof retest are nearly a full percentage point below MaTR's results. The accuracy advantage of MaTR is an aggregate indicator of a significant impact on its operational utility. Discussion of this impact requires computation of FPR and FNR (addressed later).

The best finding from Schultz's work [77], the strings classifier, has a much lower mean accuracy, and they do not include any confidence interval to describe variability in their published research. The simplicity of defeating a classifier based solely on strings [77] was a key factor in the decision to not repeat their experiment or a similar variant.

Additionally, Schultz's best structure/anomaly result has a mean accuracy of 0.8936, which is substantially lower than MaTR's. This discrepancy is most likely attributed to the small sample sizes used in their work. They state in their published research [77] that they had a limited subset of 244 PEs (206 benign and 38 malicious).

Table 19 shows the mean confusion matrix elements across the ten folds for the experiment. In the table, TP, FP, TN and FN stand for the standard true and false positives and negatives. MaTR averages only 5 total misclassifications, while the Kolter and Maloof retest has 57. Both results are impressive considering the number of samples tested.

The confusion matrix data provides the values to determine the FPR and FNR as shown in Table 20. Again, Schultz et al. do not report confidence interval data, but

Table 19. Mean confusion matrix data for MaTR and KM n-gram retest.

Method	**TP**	**FP**	**TN**	**FN**
MaTR	3,112	2	2,517	3
KM retest	3,072	14	2,505	43

Table 20. Confidence intervals for FPR and FNR.

Method	Mean	95% CI
MaTR FPR	8.73e-4	5.80e-4 — 1.17e-3
MaTR FNR	8.03e-4	4.56e-4 — 1.15e-3
KM retest FPR	5.64e-3	3.65e-3 — 7.62e-3
KM retest FNR	1.37e-2	1.23e-2 — 1.51e-2
Schultz (strings) FPR [77]	3.80e-2	*not reported*
Schultz (strings) FNR [77]	2.73e-2	*not reported*
Schultz (DLL function calls) FPR [77]	7.77e-2	*not reported*
Schultz (DLL function calls) FNR [77]	2.89e-1	*not reported*

their reported FPR and FNR appear quite different than both MaTR and the Kolter and Maloof retest results. Once again, the MaTR results for both FPR and FNR are significantly superior to those of the Kolter and Maloof retest. Furthermore, the MaTR FPR and FNR is lower than the 1% and 15-20% respectively from Tesauro's work [94], while MaTR additionally detects forms of malware other than boot sector viruses.

Finally, these FPR and FNR results illuminate a significant operational utility advantage of MaTR's methodology versus that of Kolter and Maloof. Operationally, the FPR directly relates to additional analyst workload, which is a form of resource waste as the additional samples are all non-malicious. The FNR also has operational implications, because it describes the method's inability to detect malicious samples. While neither a high FPR or a high FNR is desirable, arguably the FPR is most significant, because it has such cascading effects given the normal distortion of sampling from the non-malware and malware classes.

For example, a typical clean installation of an OS and office productivity software normally yields approximately 10,000 unique PEs, and this number will continually increase during system deployment. An advanced persistent threat (APT) may only require 1 or 2 malware artifacts to conduct effective offensive information operations on any given system. Given this estimate of a best case scenario, a 0.1% FPR yields 10 additional non-malware samples for an analyst to examine in addition to any malware samples detected. If the FPR is higher, the factor for resource waste increases linearly. This example also illustrates the value of a low FNR, because a method with a high FNR may miss the small number of malware artifacts present on a system.

4.3 Detecting Unknown Malware

This experiment tests the performance at different sensitivity levels of the trained MaTR and n-gram models from Section 4.2 and three major commercial antivirus products on a special validation set of unknown malware. This type of sample set has incredible strategic significance, because it allows for model validation (not standard training and testing) on the most challenging target, new malware undetected by most antivirus products, such as those used by APTs.

4.3.1 Methodology.

The MaTR model follows the methodology described in Section 3.4. MaTR uses exclusively PE anomaly and structural information to make its predictions. Section 4.2.1 describes the n-gram methodology. The commercial antivirus products tested use proprietary methods.

This experiment does not require traditional training or testing for machine learning. Its purpose is to validate the performance of MaTR against the n-gram model and the three commercial antivirus products on a sample set that the MaTR and n-gram models did not previously use for training or testing and that the antivirus companies likely have not analyzed.

The three commercial antivirus products tested all receive the Virus Bulletin VB100 certification for April 2011 [51]. Virus Bulletin conducts periodic independent performance evaluations of antivirus products. To receive the VB100 certification, the product must detect 100% of malware samples on the “wild list” as determined by The WildList Organization [70] and issue no false positives on Virus Bulletin’s set of clean samples using the product’s default settings.

4.3.2 Data Collection.

The validation set for these tests is a combined collection of 278 unknown malware samples discovered during incident response by anonymous organizations. Local policies restrict distribution of these discovered samples to antivirus vendors making the set “relatively unknown” for test purposes. These organizations shared this data with the stipulation that their identities remain hidden. With this agreement, duplication of these experiments with the same exact dataset is not possible, but the technique described herein has direct application to numerous other organizations that can validate these findings. Furthermore, no public database of “relatively unknown” malware exists as antivirus vendors would have previous access to the same samples to identify.

4.3.3 Experimental Design.

For the MaTR and n-gram models, this experiment uses the same features described in Section 4.2.2 and the highest accuracy classifiers from Section 4.2.6. This experiment does not involve any training or testing of any classifiers in the sense of typical classifier training and testing. It simply applies the previously trained model with the highest mean accuracy against test data (i.e., the highest, single fold accuracy from the ten folds) in Section 4.2.6 to the validation set. Table 21 shows the trained MaTR model results on test data with a decision threshold of 0.50.

The experiment has two factors: the detection method and the sensitivity threshold for the detection method. The detection method has five levels, MaTR, an n-gram

Table 21. MaTR model test data results.

Statistic	Mean	95% Confidence Interval
ROC AUC	0.999914	0.999840 – 0.999987
Accuracy	0.999166	0.999007 – 0.999325
FPR	8.73e-4	5.80e-4 – 1.17e-3
FNR	8.03e-4	4.56e-4 – 1.15e-3

model and each of the three commercial antivirus products. The sensitivity factor has three distinct levels. For these tests, the factor levels are simply the built-in low, medium, and high sensitivity levels for the commercial antivirus products. For MaTR and the n-gram model, varying the decision threshold of the classifier to 0.75, 0.50, and 0.25 for the positive class constitutes a fair and simple "standard" for low, medium, and high sensitivity levels. Typically, systems with high sensitivity tend to have lower specificity. In other words, a trade-off exists between higher detection capabilities of the positive cases and higher FPR.

These tests are side-by-side comparisons of MaTR, the n-gram method and three commercial antivirus products against the malware validation set. For consistency with prior research, these tests both adopt a standard experimental design using stratified, ten-fold cross validation. Each disjoint fold contains roughly the same number of malware samples from the validation set. During each run, a different fold functions as the test set. Additionally, the 10-fold cross validation is replicated ten times with different folds for each replication. All detection methods use identical folds across both factors to reduce experimental variance. The MaTR and n-gram models under test did not use any samples from the validation set during training or testing.

All antivirus products use default configurations, except for the sensitivity factor levels, and have updated signatures as of the date of the experiment. Each detection method runs in its own Windows 7 virtual machine with identical baseline configurations. The unknown malware samples are on a CD-ROM that each scanner examines in turn. For these tests, the commercial products all had updated scan engines and signatures as of the experiment date, March 8, 2011.

4.3.4 Results and Discussion.

Although this experiment has two factors, a full ANOVA is not necessary as the results are readily apparent. Therefore, discussion of results follows the sensitivity factor levels, including descriptions of results for all detection methods for each. Partial confusion matrices for MaTR and the n-gram model are in the Appendix.

Table 22 shows the mean scan times for each detection method (except the n-gram model as a working prototype was not available) on the entire 278 samples in the validation set. All detection methods execute on virtual machines with similar loads and access samples via CD-ROM. Although not intended as a definitive performance evaluation, the scan times demonstrate a significant execution performance advantage for MaTR versus the three commercial products. The runtime disparity is likely due to the fact that MaTR encapsulates a generic signature for malware and does not attempt to identify the specific strain of malware, a traditional requirement for antivirus products. In the table, the abbreviation `AV` refers to a commercial antivirus and the following number is the index consistent across all tests.

Table 22. Mean scan time for MaTR and commercial antivirus scanners.

Detection Method	Mean Scan Time (s)
MaTR	0.9
AV1	43.0
AV2	56.0
AV3	391.0

4.3.4.1 Low Sensitivity Level.

For test runs with the low sensitivity level, MaTR's performance is superior to the n-gram model and all three commercial antivirus products tested by a wide margin as shown in Table 23. While the best commercial antivirus product fails to reach a 50% true positive rate (TPR) against the validation set of unknown malware, MaTR correctly identifies almost 94%. The low sensitivity level of this test and the discrepancy of results highlight the significant advantage of generic malware detection over current commercial solutions. Furthermore, this advantage occurs when restricting classification decisions to static heuristic features.

MaTR performance demonstrates low variation across the ten replications of 10-fold cross validation as evidenced by the relatively narrow confidence interval. The n-gram model exhibits higher variance between runs as its confidence interval width is nearly 50% greater than MaTR's. The commercial products exhibit less consistency across folds and replications than MaTR, but they are consistent with each other. Just as MaTR's TPR is more than double the best commercial product's rate, the confidence intervals for the antivirus products are nearly double the width of MaTR's interval.

Table 23. Experiment TPR results on validation set (low sensitivity).

Detection Method	**Mean TPR**	**95% Confidence Interval**
MaTR	0.938783	0.930447 – 0.947119
n-gram	0.870504	0.858084 – 0.882922
AV1	0.456825	0.439162 – 0.474489
AV2	0.388439	0.370665 – 0.406214
AV3	0.356071	0.340914 – 0.371229

4.3.4.2 Medium Sensitivity Level.

The results from the medium sensitivity level are surprisingly similar to the low sensitivity, especially for commercial antivirus. Only MaTR, the n-gram model and AV1 exhibit any change in detections at all (see Table 24), but MaTR's performance is again statistically superior to the other methods. At this level, MaTR finds 13 more samples in the entire validation set boosting its TPR to over 98%. The n-gram solution makes substantial performance improvement as well, but maintains its higher variance evidenced by the confidence interval width. Meanwhile, AV1 detects 3 additional samples as malware overall increasing its TPR to nearly 47%. The remaining antivirus products exhibit *no* change in detections at the medium sensitivity level.

Interestingly, MaTR becomes even more consistent across the experiment folds and replications as evidenced by a 50% reduction in the width of its confidence interval. Although AV1 also increased its detection rates at the medium sensitivity level, it continues to exhibit roughly the same consistency across folds and replications. Again, the other antivirus products exhibit no change so their confidence intervals remain the same.

Table 24. Experiment TPR results on validation set (medium sensitivity).

Detection Method	Mean TPR	95% Confidence Interval
MaTR	0.985569	0.981508 – 0.989629
n-gram	0.949640	0.941552 – 0.957733
AV1	0.467606	0.449780 – 0.485431
AV2	0.388439	0.370665 – 0.406214
AV3	0.356071	0.340914 – 0.371229

4.3.4.3 High Sensitivity Level.

Tests of the high sensitivity level produce the starkest contrasts between MaTR and the commercial antivirus products as shown in Table 25. MaTR misses only 1 of the overall 278 samples in the validation set and achieves a TPR of over 99%. The n-gram solution performance increases dramatically again, but maintains is pattern of higher variance between tests. Remember, the training and testing of the MaTR and n-gram models did not include any of the malware samples in the set of unknown malware, which adds substantial strategic value to this finding. Again, MaTR demonstrates more consistent performance across folds and replications as it reduces the resulting confidence interval width by an additional 50%. The MaTR performance is statistically superior to the n-gram model and the commercial antivirus products.

AV1 detects only 1 additional sample from its medium sensitivity level. Although the best performing commercial antivirus product tested, AV1 fails to reach a TPR of 50% against the unknown malware samples. The remaining antivirus products again exhibit *no* change at the high sensitivity level.

Although a surprising result, several theories may explain why the commercial antivirus products are relatively sensitivity invariant to the unknown malware samples. Malware authors may have more thoroughly tested methods to avoid their products due to their market share or specific customers. The antivirus companies may have

Table 25. Experiment TPR results on validation set (high sensitivity).

Detection Method	**Mean TPR**	**95% Confidence Interval**
MaTR	0.996402	0.994249 – 0.998555
n-gram	0.982014	0.977638 – 0.986383
AV1	0.471216	0.453380 – 0.489053
AV2	0.388439	0.370665 – 0.406214
AV3	0.356071	0.340914 – 0.371229

made design decisions regarding trade-offs between false positives and sensitivity. Regardless of the explanation, the test results are unanticipated.

As the final planned detection test, MaTR's detection rate is more than two times greater than the best performing commercial antivirus product tested and statistically superior to the n-gram model. Against an operational validation set, this finding is significant as it shows the capability gap of commercial products versus unknown malware. These threats maneuver around commercial capabilities and rely on organizations' blind trust in antivirus products to protect them. Increasing defensive maneuver may stop or degrade these threat activities.

4.3.4.4 Union of Antivirus Products.

Due to the large disparity in results for the various sensitivity levels, additional tests examine the combined effectiveness of the three commercial antivirus products. In this case, the union of the three commercial products forms a "super" antivirus detection system. If any of the component products detect a sample, then the test considers the conglomerate system to have detected the sample.

Subject to the same experimental design, the union of all detections from commercial antivirus against the validation set still does not achieve similar performance levels as MaTR. Table 26 shows the comparative results of MaTR and the union of antivirus products at all sensitivity levels. The union performs substantially better than the individual products with a 10 to 25% TPR boost in detection performance. The same performance inconsistencies across folds and replications as seen in individual antivirus product tests is visible in the union results as well. In spite of its improvements versus the individual products at all sensitivity levels, the union still fails to appreciably challenge MaTR's (or the n-gram model's) performance.

Table 26. **MaTR and union of antivirus products TPR results.**

Detection Method	Mean TPR	95% Confidence Interval
MaTR (low)	0.938783	0.930447 – 0.947119
Union of AVs (low)	0.589894	0.571996 – 0.607792
MaTR (med)	0.985569	0.981508 – 0.989629
Union of AVs (med)	0.597090	0.579471 – 0.614709
MaTR (high)	0.996402	0.994249 – 0.998555
Union of AVs (high)	0.597090	0.579471 – 0.614709

The union of antivirus products exhibit *no* change from medium to high sensitivity levels. In this case, the additional sample detected by `AV1` in the earlier high sensitivity level test, at least one of the remaining products already found. The collective improvement from low to medium sensitivity levels for the union of antivirus products is lower than the individual improvement of `AV1` due to the same rationale.

At this point, one may conjecture that commercial antivirus products, while relatively effective at containing global or regional threats, are not nearly as effective against targeted, local threats. Traditional antivirus is helpful for containing global outbreaks of viruses and worms, but does not appear as valuable for protecting an organization from targeted threats or APTs. These threats target specific victims with custom malware that avoids detection by antivirus products and is generally not available for antivirus analysis.

The fact that commercial antivirus products detect nearly 60% of the unknown malware in this set is both a confirmation of the maliciousness of the dataset and a demonstration of antivirus shortcomings. Figure 16 shows the overlap of antivirus product detections by sensitivity level for the entire unknown set (not just test data). Although large overlaps exist, 37% of detections for the "super" antivirus product are detections by only one antivirus product (61 of 164 total detections for low sensitivity; 62 of 166 total detections for medium and high sensitivities). The relative effectiveness

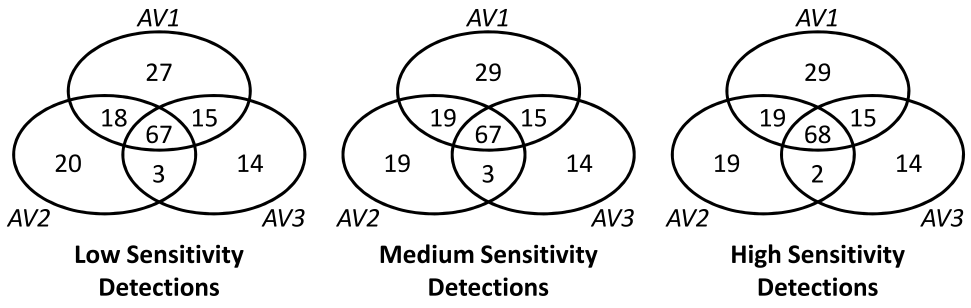
AV1
27
18
67
15
20
3
14
AV2
AV3
Low Sensitivity Detections
AV1
29
19
67
15
19
3
14
AV2
AV3
Medium Sensitivity Detections
AV1
29
19
68
15
19
2
14
AV2
AV3
High Sensitivity Detections

Figure 16. Venn diagrams showing antivirus product detection overlap for unknown samples by sensitivity levels.

of different collection techniques the antivirus companies use may account for most of the detections coming from a single product.

From the antivirus industry perspective, they cannot protect against malware they have never seen. Users expect a commercial antivirus product to not only alert the user of an infection, but to also sanitize or clean the malware as well. If threats use malware customized to avoid detection from current antivirus systems and they restrict distribution, only the victim organization has substantial opportunity to find the offending sample. Even if discovered, the victim may not desire to share the sample with malware researchers, because its an indirect acknowledgment of system or network compromise.

Given the apparent value of generic malware detection methods against unknown malware, organizations should consider investing in an in-house capability. This approach may be most appropriate for organizations with sensitive data or in extremely competitive markets.

4.4 Identifying Propagation Methods and Payloads

This series of experiments tests the performance of MaTR and compares the results to the Kolter and Maloof n-gram malware payload research [47]. MaTR tests use a much larger sampling than the Kolter and Maloof original work and assign appropriate labels based on information contained in the assigned malware names of multiple antivirus products. Automation of label assignment is simple and results in a less biased estimation of appropriate class labels than inputs collected from a single vendor. The MaTR performance statistics on like classes of malware have higher mean values than the published results from Kolter and Maloof [47]. The next two experiments examine MaTR performance against the informal antivirus industry standard for malware propagation methods [64, 86, 91] and also strategic payloads.

4.4.1 Methodology.

The MaTR model itself follows the methodology described in Section 3.4. MaTR uses exclusively PE anomaly and structural information to make its predictions. Section 4.2.1 describes the Kolter and Maloof n-gram methodology for building classifiers.

The next section describes the malware labeling method MaTR employs for training and testing. As previously discussed in Chapter II and as Kolter and Maloof attest [47], determining appropriate malware labels for machine learning is a difficult and often manual process.

As model validation, the first experiment tests the same subclasses of malware from the original Kolter and Maloof work [47], specifically mass-mailers, backdoors and viruses. These subclasses are a mixture of propagation methods and functional payloads. The next two experiments explicitly test propagation methods and strategic functional payloads. The propagation method test evaluates the MaTR classifier performance of identifying whether a given malware sample is a Trojan, worm or virus, which are the three major propagation methods the antivirus industry generally settles on. The payload test examines the MaTR classifier performance of identifying strategic payloads from a set of different malware payloads, including those not explicitly tested as a “positive” class. This payload test is therefore a demonstration of the more operationally sound application instead of merely distinguishing one payload from a sample set comprised of only three possible payloads.

4.4.2 Data Collection and Features.

The following experiments examine only 32-bit PE malware samples obtained from a download of the VX Heavens dataset [99] in April 2010. Specifically, the malware samples are Trojan, worm, and virus types as identified by the antivirus label assigned to them. Extractions of PEs initially yields 30,884 malware samples. Table 27 shows the final breakout of propagation and payload classes. Obtaining these particular class labels is a challenging problem and worthy of further discussion.

Table 27. Final malware dataset with propagation and payload sets.

Set	**No. of Samples**
Propagation (Trojan)	20,746
Propagation (Worm)	6,874
Propagation (Virus)	3,167
Payload (Backdoor)	11,077
Payload (Spyware)	7,747
Payload (Mass Mailer)	2,095
Payload (Bot)	4,003

4.4.2.1 Malware Labeling.

Automated extraction of propagation methods and payloads from malware labels is a novel approach to the supervised learning malware label problem. As Kolter and Maloof find [47], determining labels for prospective samples is a daunting task—one that typically relies on vendor classifications anyway. Ultimately, this approach allows researchers to leverage antivirus companies' expert opinions and obtain much larger samplings.

Incorporating the expert opinion of malware researchers requires harvesting relevant information from antivirus labels to assign appropriate class labels for machine learning. Recall that Kolter and Maloof generate labels by manual perusal of reverse engineering analysis reports on Symantec's website [47]. To avoid tedious manual analysis of potentially incomplete analysis reports, this research extracts and references pertinent information from antivirus product malware labels for assigning class labels.

An extraction of information from the VirusTotal website [98] for a hash list of known malware serves as a raw database from which to generate pertinent class labels. The VirusTotal information includes scan results for 41 commercial and free antivirus products, but not all products include information in a useful format for label generation for either propagation methods or payload classifiers.

Some antivirus products do not provide enough data to infer propagation and payload information. Some products that do not detect nearly all of the malware samples examined cannot provide the requisite information, because they do not have any label for the sample. Therefore, including products with low detection rates needlessly reduces the available set for training and testing. Also, if a product emphasizes *family* names and *variant* identifiers from the CARO standard, it likely has little value for label generation. Rather than deciphering each family and variant from these products, filtering out their results serves as a significant data reduction step while dramatically simplifying the problem. In these tests, filtering out these product scans leaves only labels from five antivirus products.

At this point, review of the actual antivirus product naming conventions is necessary. Guided by Occam's razor, the goal is a simple yet broad approach to categorize unknown samples without requiring manual review of analysis reports. This approach may include a small number of errors for each class, but the overwhelming majority of samples appear correct. The assigned antivirus labels are descriptive representations of a human analyst's perception.

For propagation methods, products normally use the word *Trojan* or a truncation of *Troj* or *TR* to indicate *Trojans*. *Worms* typically contain the word *worm*, while *viruses* traditionally either contain the word *virus* or list their minimal platform requirement (e.g., *Win16*, *WinNt*, *Win32*, etc.) without additional propagation information. Occasionally, antivirus products label samples as *not-a-virus*, which requires obvious handling.

Payload information is straightforward as well. *Backdoors* and bots normally include those exact words. Using the Anti-Spyware Coalition definition [4], *spyware* is "tracking software" that breaches user privacy. This definition includes obvious exact label matches, but also subclasses of *keyloggers* and the CARO standard of *pws*. In practice many labels include the abbreviation *psw*, an apparent error intended to be *pws*.

After filtering out less valuable product scan data, the antivirus product labeling data for some samples still includes multiple propagation and payload indicators. For labeling purposes, if any antivirus label indicates a propagation method or payload, the sample in question adopts all applicable labels. If one product labels a sample as a *Trojan backdoor* and another labels the same sample a *worm spyware*, then the particular sample adopts *Trojan* and *worm* for propagation labels and *backdoor* and *spyware* for payload labels. A single antivirus label may also indicate multiple payload labels.

Kolter and Maloof [46, 47] describe the interesting problems present with multi-labeled data. They illustrate this point with samples that establish a *backdoor* and log keystrokes. Rather than generating a compound class in these cases, such as *backdoor+keylogger*, they use overlapped labeling. For example, all samples with *backdoor* capabilities comprise the positive class in a one-versus-all classifier. All remaining samples compose the negative class. This approach follows the cross training method described by Boutell et al. [11] and dubbed `PT4` by Tsoumakas and Katakis [97]. The major assumption of this labeling scheme is that the compound class possesses the collective feature characteristics of each labeled class.

The labeling scheme for the following MaTR tests adopts the Kolter and Maloof approach, except the source data for the labeling is antivirus labels themselves instead of the manual review of analysis reports [47]. In these tests, the appropriate label for a sample is the presence of standard indicators for particular propagation methods or payloads. Table 27 shows the final breakout of samples and the number of each propagation method and payload. In contrast, Kolter and Maloof found label information for only 525 of their 1,651 malware samples.

4.4.3 Experimental Design.

Consistent with previous experiments, these tests adopt a standard experimental design using stratified, ten-fold cross validation using the same classification algorithm and settings, except the ensemble uses 55 trees as determined by a pilot test. Each disjoint fold contains roughly the same number of malware samples. During each run, a different fold functions as the test set while the remaining folds comprise the training set.

4.4.4 Comparison to Original Kolter and Maloof Payloads.

Table 28 shows the resulting AUC for MaTR and the original findings from Kolter and Maloof for their experiment [47]. The AUC for MaTR classification of *backdoors* is significantly superior to the original n-gram method. The remaining one-versus-all classifiers for MaTR have better mean performance than the best classifier results from

Kolter and Maloof, but not at a statistically significant level. MaTR's performance against mass mailers is statistically significant from the boosted DT result (SVM best model in original work) from Kolter and Maloof for the same class.

Table 28. Mean AUC and confidence intervals for KM payload test.

Method	Mean	95% CI
MaTR (Backdoor)	0.934058	0.928792 — 0.939325
MaTR (Mass Mailer)	0.913503	0.906645 — 0.920362
MaTR (Virus)	0.915020	0.904315 — 0.925724
KM (Backdoor) [47]	0.8704	0.8543 — 0.8865
KM (Mass Mailer) [SVM] [47]	0.8986	0.8841 — 0.9131
KM (Virus) [47]	0.9114	0.8948 — 0.9280

The Kolter and Maloof n-gram methodology has higher variance—even on the smaller dataset—than MaTR as evidenced by the confidence intervals widths. The Kolter and Maloof widths are nearly twice that of MaTR's. The confidence interval for the Kolter and Maloof *virus* classifier extends slightly past the corresponding MaTR confidence interval by 0.0023, which is less than the mean difference.

Figure 17 is the MaTR ROC plots for the Kolter and Maloof identified payloads. Comparison with the ROC plot in the original Kolter and Maloof work [47] reveals smoother plots for MaTR on all curves, an indicator of greater consistency. The MaTR *backdoor* ROC increases sharper from the origin and does not taper off nearly as pronounced as the Kolter and Maloof plot. For instance, none of the Kolter and Maloof plots (for different algorithm classifiers) reach a 0.90 TPR before hitting a 0.45 FPR, while MaTR reaches a 0.90 TPR at a 0.2 FPR. A comparison of plots for mass mailers and *viruses* yields smaller advantages for MaTR in the low FPRs (< 0.05) and higher true positive rates (> 0.90).

These tests validate the general finding of static heuristic utility for malware classification in the original Kolter and Maloof work [47]. Additionally, using readily available high-level program features instead of the resulting n-grams from high-

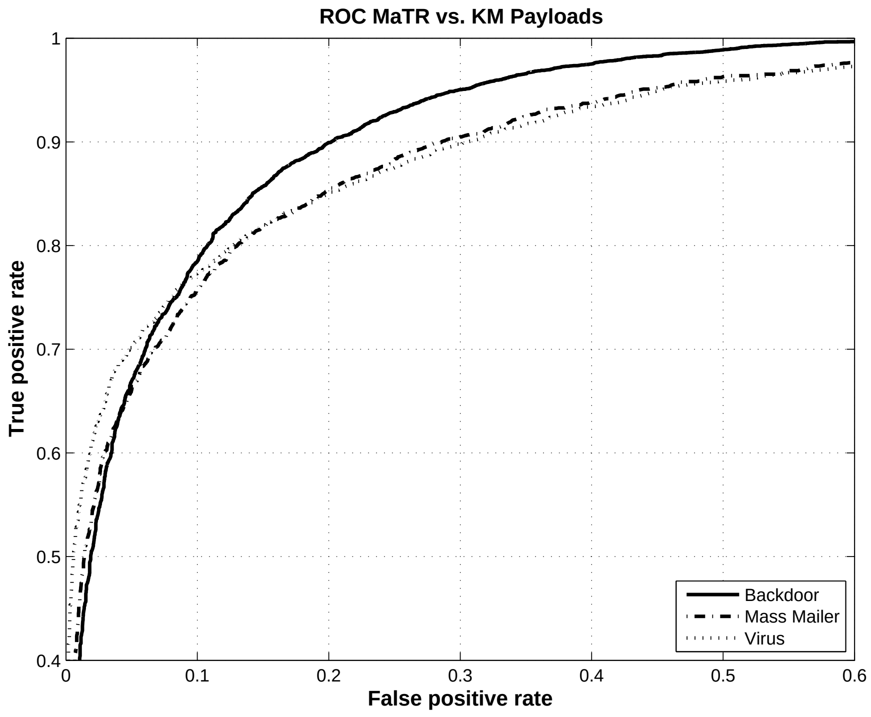

Figure 17. ROC curve of MaTR against KM payloads.

overhead computations leads to classifiers with similar or better performance. These results are consistent with previous work using the same feature source for malware detection [27] and typing [26].

The most significant observation about performance differences is the fact that MaTR substantially outperforms the n-gram method for the *backdoor* class with a mean AUC difference of 0.0637. The *backdoor* class is the most populous class in this test as it comprises nearly 68% of all samples. The degree that MaTR outperforms the n-gram classifier for the *backdoor* class and its disproportionately large class population demonstrate a significant improvement over the n-gram methodology.

Otherwise, these results are very similar to the Kolter and Maloof results [47] with subtle differences. Kolter and Maloof use a very small sample compared to MaTR, which raises questions about confidence in their findings. In their retest of the Kolter and Maloof malware detection work [46], Dube et al. [27] empirically show increased

n-gram performance with much larger sampling as Kolter and Maloof conjectured, but malware payload identification is a substantially different problem. While n-gram and MaTR features may easily find evidence of everyday malware defenses such as packing, an asymptotic limit on their performance may exist in their application to the propagation and payload problems.

In this MaTR test, the smallest class size is mass mailers with 2,095 samples, which is nearly four times as many samples as Kolter and Maloof use (525). Both methods use 10-fold cross validation which means that Kolter and Maloof tests have approximately 53 total samples in their test sets for each classifier. By comparison, MaTR has 1,640 samples in the test set for each classifier, which is a more definitive sampling.

Furthermore, the class sample populations are quite different and assumed to be reflective of the entire population. Again, given the cross validation folds, the Kolter and Maloof method likely tests with a very small sampling for the positive class. Based on ratios of the larger sampling with MaTR, their test sets for the smaller class sample populations may consist of as few as 10 samples. In contrast, MaTR test sets for the smallest class sample population (mass mailers) uses 209 samples in its test set. While the Kolter and Maloof approach in general is sound, their small sampling raises more questions about confidence in their findings versus MaTR's results.

Harvesting class information from antivirus product labels is an effective approach to the supervised learning sample labeling problem Kolter and Maloof experienced [47]. The large number of malware samples, including polymorphic and metamorphic variants, makes human review of analysis reports or similar information to glean appropriate label information infeasible.

The MaTR labeling approach also addresses antivirus vendor bias issues. An assortment of vendors may come to different conclusions on overall propagation methods and payloads. Kolter and Maloof use only information from the vxHeavens [99] and Symantec [89] websites. The vxHeavens dataset adopts the Kaspersky [42] labeling in its malware database. Kolter and Maloof expressly identify analysis reports as their sole source of label information [46, 47]. MaTR uses information from five commercial antivirus vendors in the final labeling method.

Another intangible benefit of MaTR over the n-gram approach is its use of only human understandable features. MaTR predictions result from traversing decision paths that humans can easily interpret. Kolter and Maloof include a portion of one DT [47], which they admit is not "directly useful" to human experts. While odd decision cut variables and values may exist in MaTR trees, human operators can easily discard them.

The n-gram generation overhead is another significant difference between the two methods. Kolter and Maloof do not identify how many unique n-grams their classification of the 525 samples generates. Previous tests [27] show a strong linear relationship between the number of unique n-grams and the number of samples. If the relationship holds, Kolter and Maloof likely have over 35 million unique n-grams based on their numbers [47]. While this number does not appear large, it requires a substantial amount of disk-based processing [27, 46, 47] to generate the information gains for each of the n-grams. One must not only recompute the information gains with each cross validation fold, but also harvest the unique n-grams again for each training set to avoid test contamination. MaTR's features are readily available with a one-time overhead of parser construction. MaTR's dataset does not require reconstruction for each fold, because the feature set does not change based on calculations from the training set.

4.4.5 MaTR Propagation Methods.

The purpose of the propagation test is to demonstrate MaTR performance to classify samples into one or more of the widely accepted propagation methods currently used in the antivirus industry. The following subsections detail the experimental design, results and discussion for testing MaTR against these traditional propagation methods.

The labeling for the propagation test assumes that each malware sample uses at least one of the widely accepted propagation methods found in current antivirus labels, specifically *Trojan*, *worm*, and *virus*. This experiment includes only samples that directly fall into at least one of these labeled categories as described previously in Table 27.

Table 29 shows the resulting AUC performance for the MaTR propagation classifiers. The reduced AUC for the *virus* class compared to the previous test is most likely attributable to the different classes involved in the test. While *backdoors* and mass mailers are common payloads for *Trojans* and *worms* respectively, neither pairing is synonymous. The confidence interval widths for this test are slightly narrower than MaTR's results in the previous test indicating lower variance in the results.

The apparent accuracies for the propagation classifiers (see Table 30) are lower than the AUC measures (except *viruses*) and again exhibit low variance. The *Trojan* classifier has a high FPR (43%) and a low FNR (5%). The *worm* and *virus* classifiers have low FPRs (5% and 0.9% respectively) and higher FNRs (38% and 47% respec-

Table 29. Mean AUC and confidence intervals for MaTR propagation test.

Classifier	**Mean**	**95% CI**
Trojan	0.899746	0.895569 — 0.903924
Worm	0.902971	0.897656 — 0.908285
Virus	0.890569	0.885966 — 0.895172

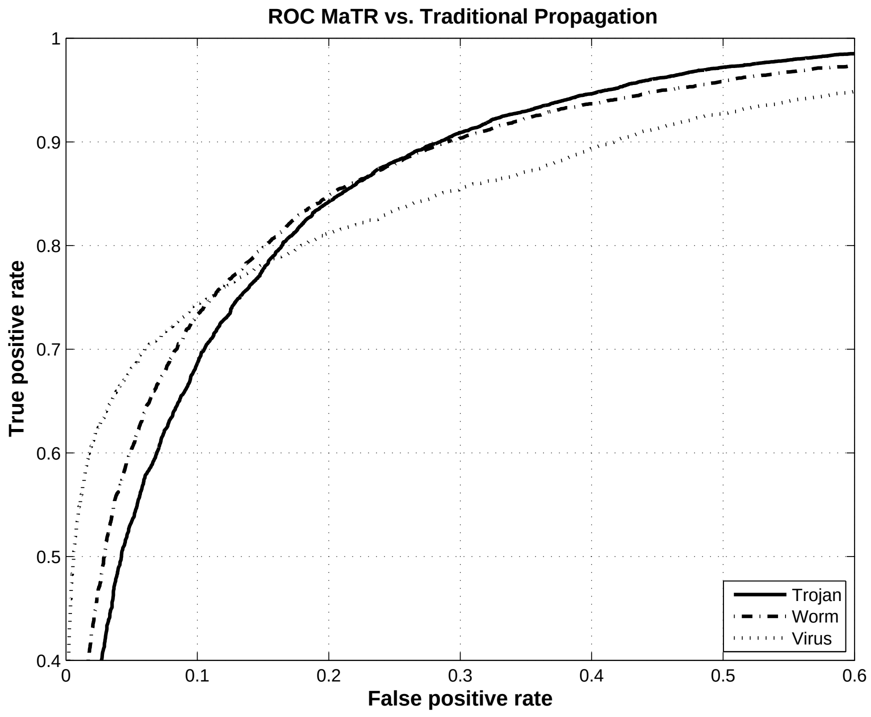

Figure 18. ROC curve of MaTR against traditional malware propagation methods.

tively). The *virus* classifier has the highest accuracy, because its low FPR applies to the largest negative class sampling which compensates for its high FNR against one of the smaller positive class samplings. These observations are also evident in the resulting ROC curves in Figure 18 with the *virus* classifier outperforming the others until it reaches a TPR of 0.75.

These tests extend the application of this data source from previous work [27] in malware detection to accurately identifying propagation methods. The payloads Kolter and Maloof identify do not align well with the malware naming convention

Table 30. Mean accuracy and confidence intervals for MaTR propagation test.

Classifier	**Mean**	**95% CI**
Trojan	0.864066	0.858389 — 0.869742
Worm	0.863110	0.859120 — 0.867101
Virus	0.937984	0.934649 — 0.941320

most antivirus companies use. This test shows how MaTR performs on the traditional malware propagation methods currently used by antivirus companies. Although not as grandiose as MaTR's detection rates [27], this additional information provides useful context for dynamic risk assessment. For instance, given a MaTR detection of a *worm* or *virus*, a relatively high likelihood exists that without an immediate response the infestation may spread causing substantially more effort to contain. A *Trojan* detection can indicate an APT purposefully traversing the company network or a user inadvertently downloading "adware".

Depending on current operations, the operations climate, and data sensitivity of the system in question, containment may be less significant to potential data exfiltration by an APT. While additional threat information, such as payloads, is valuable, identifying propagation methods is a significant foundation for threat response, specifically to level of containment effort.

The lopsided FPR and FNR fortuitously occur in advantageous application areas. The *worm* and *virus* classifiers correspond to the two classes most likely to spread like wildfire through the company network. These two classifiers are also the two with the lowest number of false positives, which means alerts should not condition users to ignore them as with other security products. The high FPR for the *Trojan* classifier is associated with the much smaller negative class set, while its low FNR implies that it should not miss many *Trojan* predictions.

This additional threat information comes with no manual analysis of the sample or overhead of more resource-intensive methods. It also provides human operators with actionable information regarding the threat as described above. As with any detection method, manual confirmation is necessary as all methods have the propensity to generate false positives. In many cases of a malware detection, re-imaging the system is likely the best option available even over cleaning by an antivirus product. Symantec even recommends this approach for some malware infections [90].

These findings show utility for prioritization of slower runtime, but more intensive, methods to determine threat information. While dynamic heuristic methods generally have high accuracy, the runtime overhead required is typically incomplete or operationally infeasible for analyzing the complete set of files. By first reducing the set to the most suspect, these methods become more operationally viable and valuable to the organization.

One can envision these issues in the following scenario of a workstation with 30,000 unique executables, which may be a conservative estimate. If samples run in a sandbox environment for five seconds each, it takes 1.74 days to complete a scan of all executables on the system. At this point in the scenario, three observations are readily apparent. First, measuring scan time in days is not particularly a winning strategy. Second, the assumption of a five second (or any fixed time) observation period being sufficient is flawed as many malware samples, such as the Michelangelo virus [88], do not execute until after a specific time period or event occurs. Third, sandbox execution may fail due to specific dependencies not being met as Szor describes in detail [91].

4.4.6 MaTR Strategic Payloads.

This test demonstrates MaTR's ability to identify a few strategic value malware payloads in an operational manner. Assuming previous identification of a generic malware detection, this test demonstrates a series of subsequent decisions regarding the payload of the discovered artifact. The specific payloads tested are *backdoors*, *spyware* (including *keyloggers*), and malicious *bots*. The following subsections describe the experimental results and discussion for this test against strategic payloads.

This experiment uses the same experimental design of the propagation experiment. The labeling for the payload test simply examines whether or not the samples have the given functionality. The dataset includes explicitly labeled samples for three major malware payloads as well as generically labeled unknown payload samples for more direct operational employment. In other words, this experiment includes all malware samples from the collection regardless of their labeled payload. A total of 8,057 samples with other types of payloads are included in stratified quantities in the training and test sets.

Kolter and Maloof do not expressly state whether or not they use their other unknown samples (1,126 unknown malware payload samples) in their tests [47]. They most likely omit this unknown subset, because their labels came from manual assessment of analyst reports. During their assessment, they cannot speculate labels for remaining malware samples, because it leads to a high likelihood of having positive class samples (any of the three classes they test) in the negative unknown subset. Conversely, the labeling used for the MaTR tests has a lower likelihood of positive class samples being present in the negative class, because of the selection of antivirus labeling schemes and the strong agreement between them.

Tables 31 and 32 are the respective AUC and accuracy for the MaTR strategic payload tests. While the results are generally lower than in the propagation tests, they are operationally relevant as the classifiers identify these positive payloads while including a diverse variety of other payloads in each negative class set. The *bot* classifier performance exceeds all others with an impressive 96.6% prediction rate.

Table 31. Mean AUC and confidence intervals for MaTR payload test.

Classifier	**Mean**	**95% CI**
Backdoor	0.893490	0.889489 — 0.897490
Spyware	0.851004	0.847215 — 0.854794
Bot	0.960844	0.957540 — 0.964148

Table 32. Mean accuracy and confidence intervals for MaTR payload test.

Classifier	Mean	95% CI
Backdoor	0.833863	0.829968 — 0.837758
Spyware	0.834477	0.830592 — 0.838363
Bot	0.965710	0.963464 — 0.967957

The resulting ROC curve (see Figure 19) for the payload classifiers supports the quantitative data. The *backdoor* classifier FPR and FNR are 7.9% and 32.2% respectively. The *spyware* classifier has a 5.2% FPR, but a 50.5% FNR. The best performing classifier of the set, the malicious *bot* classifier, has a 0.8% FPR and 21.4% FNR. All classifiers' low FPRs correspond to the much larger negative classes, which is an advantageous characteristic. The relatively high FNR indicate that they misclassify many instances of the positive class, especially the *spyware* model that misses over half of the positives.

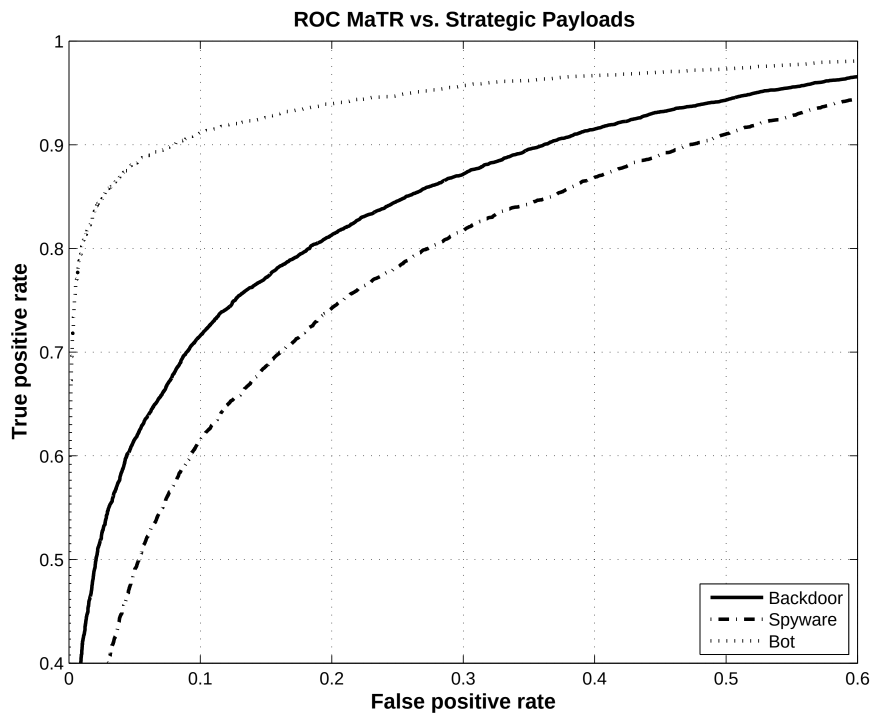

Figure 19. ROC curve of MaTR against strategic malware payloads.

These tests extend the application of this data source from previous work [27] in malware detection to identifying likely payload. Since these tests include other additional payloads, these findings are operationally significant, because they demonstrate the ability to deduce target payloads from a wide variety of other, non-enumerated payloads using only static heuristics. These findings are significant, because they demonstrate MaTR's ability to provide additional threat information without human analysis or time-consuming dynamic analysis methods.

The following scenario demonstrates the collective findings. Company ABC uses an excellent commercial antivirus product, which their chief competitor, XYZ, discovers. Threat XYZ has strong motivation to steal ABC's intellectual property and makes a small investment to modify existing malware to avoid detection by ABC's antivirus product. XYZ establishes a strong presence on ABC's network and conducts a variety of offensive information operations for over two years. Company ABC suspects this activity as XYZ has mysteriously become more competitive in the market, but rarely, if ever, discovers any indications to validate their theory.

With a technology similar to MaTR, ABC has a very strong chance of detecting XYZ on their network early [27]. Section 4.3 shows how MaTR detects 98.6% of unknown malware while three major commercial antivirus products combine to detect less than 60%. As XYZ has multiple tools on ABC's network, ABC must generate a prioritized response plan to remediate the newly discovered tools. At this point, MaTR can quickly provide ABC with additional threat information allowing ABC to construct a rational response plan. ABC may choose to address self-propagating malware first for containment purposes and next pursue *backdoor* and *spyware* remediation to stop the loss of intellectual capital. If operational circumstances dictate otherwise, ABC may select a more appropriate response to meet its unique needs.

Based on the propagation and payload tests, MaTR shows the greatest potential for identifying fast-spreading *worms* and *viruses* with various payloads. These identifications have the lowest number of false positives, but the highest rates of generating false negatives. Exploration of additional static features may provide additional insight into these predictions.

4.5 Mapping to Endsley Situation Awareness Model

Threats routinely use malware to conduct offensive information operations, making cyberspace SA regarding malware a significant goal for Information and Communication Technologies (ICT) operators. Bejtlich [8] states that achieving cyberspace SA is a method to thwart APT operations. As previously described with Christodorescu and Jha [14, 15] and as Section 4.3 demonstrates, threats can easily evade commercial antivirus products by altering existing malware and exercising restraint when considering the number of targets for custom malware.

MaTR experiment results demonstrate significant potential to enhance an operator's ability to achieve SA in cyberspace, because it provides timely and higher accuracy malware detection and additional threat information using non-instruction-based static heuristics than current research and commercial methods. MaTR malware detection results are statistically superior to a popular n-gram method [46, 47]. The MaTR identification of backdoor payloads is statistically superior to similar work using an n-gram method [47] with higher mean performance for the remaining two payloads from the original n-gram work.

While MaTR does not provide context, it does provide operators information which is a "perception of relevant elements in the environment, as determined by system displays..." as Endsley [33] describes, which is the foundation for establishing SA [33]. From the operator perspective, timely additional threat information is a critical enabling capability in complex, dynamic, high-threat environments. This section describes operator SA implications based on MaTR's performance using the Endsley model.

Figure 20 shows a logical application of the primary portion of the Endsley SA model [33] to the MaTR outputs. Endsley lists several examples of the first step of achieving SA, Level 1, specifically "to perceive the status, attributes, and dynamics of relevant elements in the environment" [33]. In one of her examples, drivers need to know location and state information for their own vehicle, other vehicles and obstructions. "Relevant elements" in cyberspace regarding malware require a high level of detection (i.e., state information) to first perceive the threat and also include propagation methods and payloads, which are both attributes further describing the malware element. Other attributes are possible, such as attribution information. Therefore, all MaTR outputs are pertinent to the operator establishing Level 1 SA. Collectively they provide a much clearer view of the threat environment allowing for simpler operator acquisition of Level 2 SA.

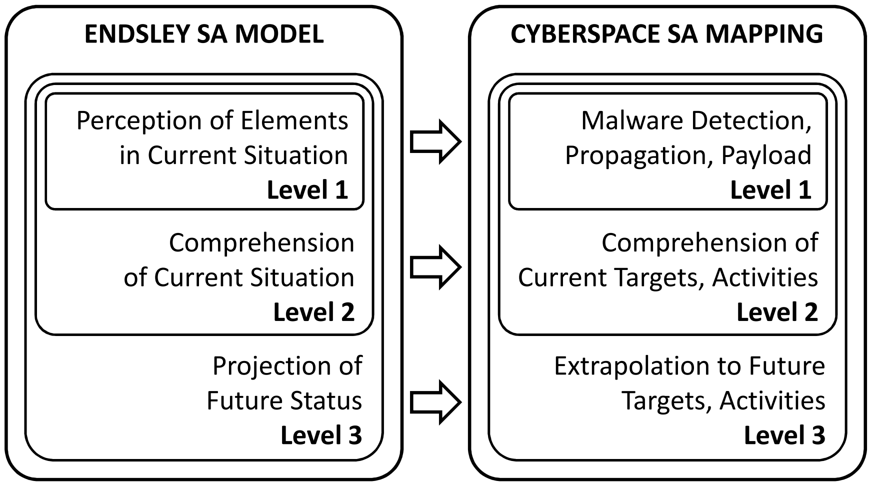
ENDSLEY SA MODEL
Perception of Elements in Current Situation
Level 1
Comprehension of Current Situation
Level 2
Projection of Future Status
Level 3
CYBERSPACE SA MAPPING
Malware Detection, Propagation, Payload
Level 1
Comprehension of Current Targets, Activities
Level 2
Extrapolation to Future Targets, Activities
Level 3

Figure 20. Mapping of Endsley situation awareness model (adapted from [33]).

The presence of certain malware payloads on systems with access to sensitive information is a strong indicator of threat interest and intention. The identification of such systems is another example of a pertinent Level 1 SA element, but this information should be readily available to operators or their organizations, because it is under their direct control. In the projection of the Endsley SA model to cyberspace regarding malware threats, comprehending current targets and threat activities is an example of Level 2 SA. Extrapolating future targets and activities is an example of Level 3 SA.

The following scenario is a continuation of the scenario previously introduced in Chapter I. The network diagram in Figure 1 shows three major business divisions and a sampling of endpoint systems within each. The figure also indicates endpoints infected with malware in each division identified by a generic detection capability and not as a result of a commercial antivirus product.

The following discussion assumes that the hypothetical system interface clearly portrays the same information to the operator. One method to implement this interface capability is to have local copies of the detection classifier report alerts to a central server along with the infected system's address. The central server can then map these alerts to the current network diagram.

4.5.1 Achieving SA with Detection Information.

Interpreting the above situation via Endsley's model [33], the human operator is likely overwhelmed for multiple reasons. The model's system factors (system and interface design, stress, workload, complexity and automation) all have negative impacts on the operator SA. Individual factors, such as ability, experience and training, may not compensate for the negative influences of the system factors. The limitations on working memory and attention make this scenario even more complicated for the operator. A brief exposition of the system factors follows.

First, the detection system and interface do not directly provide the necessary information to reach Level 2 SA. The operator must manually search for corroborative elements to support their initial perception of the environment. Specific elements the operator must determine regard the nature of the threat, which may require human analysis after sample collection. Matching the available known elements from the situation to available schemata is difficult as the known elements may correspond to multiple schemata related to malware. There are several major classes of malware payloads and three major propagation methods, but nearly all are associated with different scripts. The cyberspace "big picture" is not readily apparent for human comprehension as the alerts may be adware or even empty payloads which are common for computer viruses [91]. Operating on the incomplete or erroneous SA may adversely affect operator decision making processes minimizing effectiveness for all experience levels.

The operator may experience increased stress in tense dynamic situations when the system presents alerts. The operator feels compelled to act on the alert, but he has limited visibility into the problem. Due to ICT complexity, the operator may not have other automated means of assistance to reach Level 2 SA, which further contributes to stress. As the operator struggles, the additional stress may cause "premature closure" [33], where the operator reaches a premature decision based on limited Level 1 SA. When this happens, operator bias may prevent achieving a real understanding of the situation. At this point, the tactical stress effectively degrades the operator decision making processes.

With only detection information, the operator workload necessarily increases. The operator must strive to acquire the additional pertinent elements related to functionality in order to achieve Level 2 SA. If these elements are only available via slower manual processes, reaching the next level may not be available within time-sensitive tactical situations.

The complexities of the above scenario are readily apparent. The state data for even small networks is so large that automated IDSs cannot maintain it. The human operator also cannot visualize the state data, because no tools presently provide this capability. In addition to the complexity of the automated elements, the threat itself is another complex element changing actions based on its perception of the environment and after accomplishing assigned tasks.

Finally, the automation afforded by the detection capability provides incomplete information and increases operator workload while simultaneously increasing stress. As a result, the automation likely causes more stress and workload similar to that of current IDSs, which are notorious for flooding users with false alerts. The alert flood serves to condition the operator to ignore the system. Running the system with lower sensitivity, the operator interfaces with it less often causing the system to become ineffective. Section 4.5.2 describes how MaTR outputs of relevant threat information impacts the operator's ability to achieve SA.

For a practical application of the Endsley SA model, what can the operator deduce from this scenario with only detection information? Most notably, the operator can observe the number of infected hosts in each business division. This information is not actionable, because it does not provide a solid foundation for Level 2 SA to justify specific actions. With only the detection information, the operator must treat all infected systems as the worst local situation possible, which is impractically resource intensive. The number of infected hosts combined with the known sensitivities of particular business divisions may be enough information to match a particular schemata and trigger effective scripts, but only in rare cases.

4.5.2 Achieving SA with MaTR.

Based on MaTR outputs, the available network diagram provides a sufficient description of elements to enable the operator to achieve Level 2 and Level 3 SA (see Figure 21). In the previous scenario, all threats look identical, but MaTR can distinguish between them. A discussion of the same system factors follows.

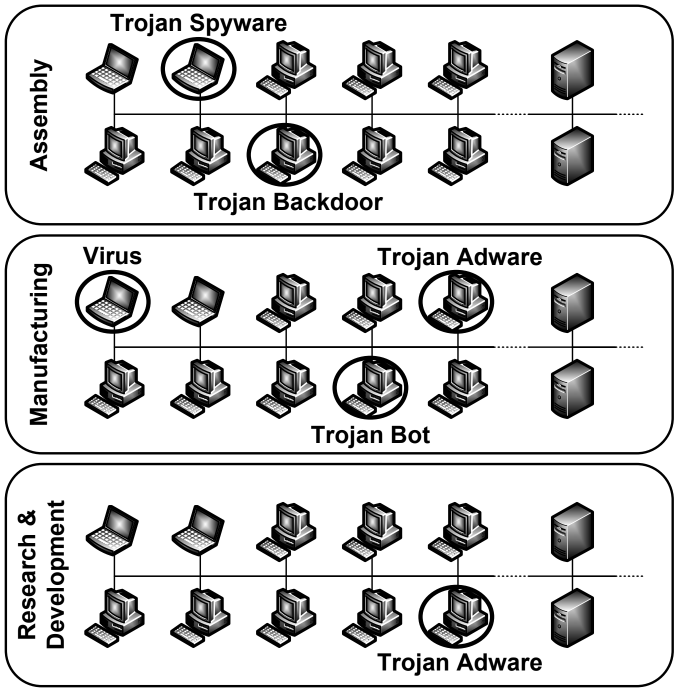

Figure 21. Situation awareness indicators afforded with additional threat information.

First, the system provides additional threat information not directly available with current technology allowing the operator to more accurately assess malware behavior and capabilities in the environment. End users researching functionality of specific malware names provided by antivirus products is an example of unavailability of the threat information at alert time. With the additional threat information at alert time, an operator with minimal training can effectively triage the network and form an appropriate response plan to address current threat activities and targets (Level 2 SA). In this scenario, the most significant threats are the backdoor and spyware located in the assembly division. These tools allow for a wide range of effective offensive information operations. The identification of the propagation methods and payloads allows for simple schemata matching with appropriate scripts. Response scripts associated with the matched schemata of the backdoor and spyware include re-imaging the infected systems followed by immediate compulsory password changes.

The next most significant threat is in the manufacturing division. The operator's Level 2 SA enables him to reason whether the bot or the virus is the most significant threat. At this point, the operator may have knowledge of other Level 1 elements, such as the manufacturing division's sensitivity to system availability. These additional out-of-band elements are important as well and they apply equally to the previous scenario. The least significant threats are the adware threats.

Furthermore, from a stronger Level 2 SA, the operator can now project threat activities and targets (Level 3 SA). The operator may deduce that the threat may be preparing for the final stages of production of a similar system as their presence appears strongest on the assembly division. Perhaps their competitor has already exfiltrated necessary data from the research and development division and manufacturing divisions and now seeks assembly secrets to ensure their success. Projection of specific future threat activity include the use of stolen user credentials to maintain access to the network or to gain access to additional information and the exfiltration of large amounts of data.

While undoubtedly stressful to discover significant threat activity, MaTR does not add to operator stress as in the previous scenario. In the previous scenario, the detection capability is similar to a fire alarm in a large building. It creates more questions than it answers. Where's the fire? Is this the safest escape route? Is the fire department responding? MaTR likely reduces stress by providing strong indicators of meaningful data especially given its low FPR from Section 4.2. As a

DT implementation, it is feasible for the operator to examine its "rationale" for the alert, because its features are directly human understandable.

As the operator workload increases due to perception of the situation, the additional workload focuses on addressing the threat, not necessarily searching for additional elements to confirm theories of threat capability. MaTR outputs result in less unknowns and all without manual analysis of the detected malware. The outputs focus on providing clarity to the operator of an otherwise extremely complex and dynamic system. MaTR provides this additional detail significantly faster than commercial antivirus products as shown in Section 4.3.

Endsley [33] describes automation as a potential benefit or hindrance to SA. She depicts how automation is a common reason for *degraded* SA, especially for periods when the system is not available to the user and the user has developed a strong reliance upon it. The case with MaTR matches an exception, because it "reduces unnecessary manual work and data integration to achieve SA" [33].

4.6 Summary

MaTR demonstrates superior performance to n-gram detection research. Reconstruction of the Kolter and Maloof experiment in malware detection [46, 47] allows for a true apples-to-apples comparison using the same PE sources. Both methods use identical folds for cross validation and compute metrics in the same manner. Using the set of unknown malware PE files as a separate validation set, MaTR again outperforms the n-gram detection method.

Against unknown malware, MaTR outperforms three major commercial anti-virus products. The commercial products do not appreciably increase performance at their highest sensitivity levels, while MaTR detection rates exceed 90%. Collectively, the three commercial products fail to detect 60% of the unknown malware samples.

For propagation method and payload identification, MaTR has higher mean performance than the reported n-gram results reported by Kolter and Maloof [47]. The MaTR results in the payload tests are pessimistic as they include additional classes of malware other than those explicitly tested as positive classes, which is more representative of the true malware population and more operationally applicable. Future work includes performing another exhaustive comparison as with their detection work to ensure that both methods train with the same class labels and folds.

Section 4.5 describes how simple detection information alone does not provide adequate SA, even for experienced operators. The detection information prompts more questions than it answers causing it to ironically *decrease* SA by complicating system factors. MaTR outputs, while elements of Level 1 SA, provide significantly greater threat information enabling the operator to achieve Level 2 and even Level 3 SA. The focused visibility into the threat environment that MaTR provides the human operator dramatically exceeds awareness afforded by simple detection alone.

In more concrete terms, MaTR detection accuracy of 99.9% reported in Sections 4.2 and 4.3 is superior to the best performing static heuristic research and a combination of three commercial antivirus products. MaTR demonstrates the ability to identify unknown threats with 98.6% accuracy (Section 4.3) where traditional defenses fail to detect 60%. While not as spectacular as its detection rates, MaTR also provides propagation information with 86+% accuracy and identifies strategic payloads from all malware payloads with an accuracy of 83+%.

V. Conclusion

In 2010, Symantec saw 286 million new malware threats, a 19% increase from the rate of new threats in 2009 [24]. The same article quotes Symantec's chief executive officer from February 2011, when he said that "traditional antivirus scans 'long ago failed to keep up'". Worse yet, operators have little visibility into the complex and hyper-dynamic state of Information and Communication Technologies (ICT) and thus have minimal SA.

MaTR uses standard machine learning algorithms and traditional anomaly and structure information to achieve detection accuracy of 99.9% on test sets composed of non-malware and malware. This accuracy is a statistically significant improvement over related work. In validation tests against a set of unknown malware, the same trained model again demonstrates statistically significant superior performance over both a reconstruction of tests from leading related work and three commercial antivirus products combined.

The major goal for MaTR is to provide enhanced cyberspace SA for operators, where they traditionally have minimal visibility. By restricting the model to intelligible high-level information, the model can provide human operators with justification for its predictions unlike other models. The solution is better suited for tactical employment due to its low runtime performance than other more exhaustive techniques that use tracing, generate disassembly and control flow graphs.

Previous research applications of machine learning to malware detection predominantly center around the analysis of n-grams, which in spite of their widespread employment are relatively meaningless short byte string sequences. Kolter and Maloof [46, 47] admit the difficulty of assigning meaning to many of the most relevant n-grams. These n-grams may be stronger indicators of the presence of certain strings and data offsets than instructions as many generally believe.

Practitioners must be cautious when relying on n-grams as the primary feature source as it is completely within the attacker's realm of control and tactical modifications in this area are low cost and low effort. The risk associated with investing financial and technical resources into an unexplainable and unjustifiable defense against an adaptable foe is intuitively high. Making strategic decisions to use such operationally untested solutions can leave an organization at the mercy of trivial changes in attacker tactics. This observation is analogous to an adversary developing a \$100K missile that defeats a \$1B stealth bomber.

Several other techniques require elusive instruction-level information, which is difficult to validate and time intensive to collect. Collection methods include dynamic debugger tracing and static unpacking and disassembly. Malware has sufficient protections against full automation of both methods as malicious logic often includes debugger detection and instruction obfuscation capabilities. This research does not attempt to gather this elusive data for classification purposes.

This research already demonstrates substantial technological gain for the USAF as it directly and immediately addresses at least four of the eight AFRL identified FLTCs in cyberspace, especially "Assured Operations in High Threat Environments". In spite of the described weaknesses in some of these classifiers, any of these models provides a great capability leap for the USAF, Department of Defense, and the US government. The USAF needs a robust malware detection model that can provide additional threat information in order to survive against adaptable foes in an increasingly dangerous cyberspace.

5.1 Research Contributions

This research demonstrates multiple engineering advantages for static heuristic malware detection and classification using only traditional anomaly and structural data as features. The resulting classifiers for malware detection, propagation methods and payloads are simpler to construct and perform at similar or superior levels to published results for leading research using static heuristics [46, 47]. Against a validation set of unknown malware, MaTR significantly outperforms n-gram classifiers built according to [46, 47] and three commercial antivirus products—even when combining their detection capabilities.

MaTR joins the original Kolter and Maloof [47] work as a rare application of static heuristics to providing threat information in addition to simple detection. MaTR performs better than their n-gram classifiers for identifying malware payloads. This research also reports MaTR performance for predicting the propagation methods currently used by the antivirus industry and payloads in an operational manner.

This work also makes a unique contribution by applying the Endsley [33] SA model to MaTR capabilities and projecting it as a positive impact on cyberspace SA. Simple detection alone more likely hampers operators from reaching higher levels of SA, but MaTR's focused outputs address operator information requirements by providing relevant threat information. A simple scenario illustrates the value of the additional threat information that MaTR provides.

Lastly, this work has resulted in three publications to date including one thesis [25, 26, 61]. Future publications include one journal submission currently in second review [27].

5.2 Future Research Recommendations

Recommendations for future research include continuing evaluation of MaTR-like systems versus other static heuristic methods. In particular, reconstructing the Kolter and Maloof [47] original work with the same labeling method as used in Section 4.4 may reveal additional advantages for non-instruction-based, static heuristic classifiers.

The identification of other disparate, SA-focused cyberspace sensors will serve to increase the robustness and capabilities of this research. Systems like MaTR can be one of a set of sensors to fuse together in order to provide cyberspace operators greater capabilities to achieve SA. Other potential sensors include network and log classifiers.

Extensions to MaTR include examinations of instruction-level data and dynamic heuristics as other sources of Level 1 SA elements and as means to further validate MaTR performance. Investigating data fusion of these systems may provide immediate operator feedback while simultaneously instantiating confirmation with slower, but potentially more accurate sensors. These approaches may further increase operator confidence levels in overall system predictions.

Appendix A: Confusion Matrices

A.1 Modeling Approach and Best Classifier Pilot

Tables 33, 34, 35 and 36 are the confusion matrix data for the detection problem from Section 3.3.3.1.

Table 33. LDF confusion matrix test results for detection problem.

		Predictions	
	Class	NM	M
Actual	NM	$2,165$	178
	M	248	805

Table 34. DDA confusion matrix test results for detection problem.

		Predictions	
	Class	NM	M
Actual	NM	$2,165$	178
	M	278	775

Table 35. FNN confusion matrix test results for detection problem.

		Predictions	
	Class	NM	M
Actual	NM	$2,237$	106
	M	80	972

Tables 37, 38, 39 and 40 are the confusion matrix statistics for the detection problem from Section 3.3.3.1.

Tables 41, 42, 43 and 44 are the confusion matrix data for the direct typing problem from Section 3.3.3.1.

Table 36. DT confusion matrix test results for detection problem.

		Predictions	
	Class	NM	M
Actual	NM	2,301	42
	M	38	1,014

Table 37. LDF test confusion matrix statistics for detection problem.

Statistic	NM	M
Producer Accuracy	0.9239	0.7647
Consumer Accuracy	0.8973	0.8186
Omission Error	0.0761	0.2353
Commission Error	0.1027	0.1814
Kappa	0.7013	

Table 38. DDA test confusion matrix statistics for detection problem.

Statistic	NM	M
Producer Accuracy	0.9241	0.7363
Consumer Accuracy	0.8864	0.8133
Omission Error	0.0759	0.2637
Commission Error	0.1136	0.1867
Kappa	0.6780	

Table 39. FNN test confusion matrix statistics for detection problem.

Statistic	NM	M
Producer Accuracy	0.9547	0.9235
Consumer Accuracy	0.9653	0.9016
Omission Error	0.0453	0.0765
Commission Error	0.0347	0.0984
Kappa	0.8724	

Table 40. DT test confusion matrix statistics for detection problem.

Statistic	NM	M
Producer Accuracy	0.9821	0.9639
Consumer Accuracy	0.9837	0.9603
Omission Error	0.0179	0.0361
Commission Error	0.0163	0.0397
Kappa	0.9450	

Table 41. LDF confusion matrix test results for direct typing problem.

		Predictions			
	Class	NM	BD	T	V
Actual	NM	2,116	33	155	38
	BD	67	343	221	40
	T	28	55	94	9
	V	13	15	80	87

Table 42. DDA confusion matrix test results for direct typing problem.

		Predictions			
	Class	NM	BD	T	V
Actual	NM	2,165	178	0	0
	BD	174	498	0	0
	T	65	121	0	0
	V	39	156	0	0

Table 43. FNN confusion matrix test results for direct typing problem.

		Predictions			
	Class	NM	BD	T	V
Actual	NM	2,279	52	0	11
	BD	62	587	0	22
	T	29	143	0	14
	V	23	51	0	121

Table 44. DT confusion matrix test results for direct typing problem.

		Predictions			
	Class	NM	BD	T	V
Actual	NM	2,292	36	10	5
	BD	24	536	92	20
	T	8	74	83	21
	V	6	21	29	140

A.2 Model Parameter Assessment Pilot

Table 45 is the mean confusion matrix for the model parameter detection test from Section 3.3.3.2. Table 46 is the mean confusion matrix for the model parameter typing test from Section 3.3.3.2.

Table 45. Detection classifier results mean confusion matrix.

		Predictions	
	Class	NM	M
Actual	NM	2,653	14
	M	12	1,321

Table 46. Malware type classifier results mean confusion matrix.

		Predictions						
	Class	BD	DW	TJ	PS	W	DR	V
Actual	BD	1,419	112	131	85	69	54	19
	DW	121	690	109	43	28	43	13
	TJ	151	121	241	45	38	38	22
	PS	120	51	55	237	22	28	8
	W	89	31	46	20	163	14	18
	DR	82	60	49	33	18	139	6
	V	36	20	42	12	20	8	178

A.3 Detecting Unknown Malware

Table 47 is the partial confusion matrix data for MaTR and the n-gram methods on the relatively unknown malware validation set. This test does not include a negative class and therefore has only partial results (no true negatives or false positives) to report in the confusion matrix.

Table 47. Partial confusion matrix data for MaTR and *n*-gram detection methods on unknown malware validation set (no negative class).

Method (Sensitivity)	TP	FN
MaTR (low)	26.10	1.70
n-gram (low)	24.20	3.60
MaTR (medium)	27.40	0.40
n-gram (medium)	26.40	1.40
MaTR (high)	27.70	0.10
n-gram (high)	27.30	0.50

A.4 MaTR Additional Threat Information

Tables 48, 49 and 50 are the mean confusion matrices for MaTR propagation tests from Section 4.4.

Table 48. Mean confusion matrix data for MaTR on KM-defined payloads [47].

Class	TP	FP	TN	FN
Backdoor	1,055	142	316	53
Mass mailer	97	15	1,341	112
Virus	192	24	1,225	125

Table 49. Mean confusion matrix data for MaTR propagation methods.

Class	TP	FP	TN	FN
Trojan	1,981	276	371	94
Worm	424	109	1,926	264
Virus	169	21	2,384	148

Table 50. Mean confusion matrix data for MaTR payloads.

Class	TP	FP	TN	FN
Backdoor	752	157	1,824	356
Spyware	384	120	2,193	391
Bot	315	20	2,668	86

Bibliography

[1] Abou-Assaleh, Tony, Nick Cercone, Vlado Keselj, and Ray Sweidan. "N-gram-based Detection of New Malicious Code". *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 41–42. IEEE, New York, USA, 2004.

[2] Albus, J. S. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)". *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, 1975.

[3] Anderson, Ross. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, New York, NY, USA, 2001.

[4] Anti-Spyware Coalition. "Definitions and Supporting Documents: Glossary", 2007. URL www.antispywarecoalition.org/documents/2007glossary.htm. Accessed May 5, 2011.

[5] Arnold, William and Gerald Tesauro. "Automatically Generated Win32 Heuristic Virus Detection". *VIRUS*, 51, 2000.

[6] Avira Soft SRL. "AVIRA Malware Naming Conventions", 2009. URL www.avira.ro/en/virus_information/malware_naming_conventions.html. Accessed Sept. 18, 2009.

[7] Bailey, M., J. Oberheide, J. Andersen, and Z. M. Mao. "Automated Classification and Analysis of Internet Malware". *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection*, 178–197. Springer, 2007.

[8] Bejtlich, Richard. "What APT is (And What it Isn't)". *Information Security*, 12(6):20–24, 2010.

[9] Bergeron, J., M. Debbabi, M. M. Erhioui, and B. Ktari. "Static Analysis of Binary Code to Isolate Malicious Behaviors". *8th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. 1999.

[10] Bontchev, Vesselin. "Current Status of the CARO Malware Naming Scheme", 2009. URL www.people.frisk-software.com/~bontchev/papers/pdfs/caroname.pdf. Accessed Sept. 15, 2009.

[11] Boutell, M. R., J. Luo, X. Shen, and C. M. Brown. "Learning multi-label scene classification". *Pattern Recognition*, 37(9):1757–1771, 2004.

[12] Carrera, E. "pefile", 2008. URL code.google.com/p/pefile/. Accessed Oct. 21, 2008.

[13] Chapman, P., J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. "CRISP-DM 1.0: Step-by-step data mining guide", 2000. URL www.crisp-dm.org/CRISPWP-0800.pdf.

[14] Christodorescu, Mihai and Somesh Jha. "Static Analysis of Executables to Detect Malicious Patterns". *Proceedings of the 12th USENIX Security Symposium*, 169–186. USENIX Association, Berkeley, CA, USA, 2003.

[15] Christodorescu, Mihai and Somesh Jha. "Testing Malware Detectors". *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 34–44. ACM, Boston, MA, USA, 2004.

[16] Christodorescu, Mihai, Somesh Jha, Sanjit Seshia, Dawn Song, and Randal Bryant. "Semantics-aware Malware Detection". *IEEE Symposium on Security and Privacy*, 32–46. Citeseer, 2005.

[17] Christodorescu, Mihai, Nicholas Kidd, and Wen-Han Goh. "String Analysis for x86 Binaries". *ACM SIGSOFT Software Engineering Notes*, 31(1):95, 2006.

[18] Cohen, William W. "Fast Effective Rule Induction". *Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning*, 115–123. Morgan Kaufmann, San Francisco, CA, USA, 1995.

[19] Collberg, Christian, Clark Thomborson, and Douglas Low. "A Taxonomy of Obfuscating Transformations", 1997.

[20] Dahm, Werner J. A. "Overview of the Air Force Science & Technology Enterprise", Nov. 2009. Briefing to Air Force Institute of Technology.

[21] Dictionary.com. "Dictionary.com Unabridged", Jan 2010. URL dictionary.reference.com/browse/occamsrazor.

[22] Dillon, W. R. and M. Goldstein. *Multivariate Analysis: Methods and Applications*. Wiley, New York, NY, USA, 1984.

[23] Dinaburg, Artem, Paul Royal, Monirul Sharif, and Wenke Lee. "Ether: Malware Analysis via Hardware Virtualization Extensions". *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 51–62. ACM, New York, NY, USA, 2008.

[24] Drew, Christopher and Verne G. Kopytoff. "Deploying New Tools to Stop the Hackers", 2011. URL www.nytimes.com/2011/06/18/technology/18security.html. Accessed Jun. 19, 2011.

[25] Dube, T., R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers. "An Investigation of Malware Type Classification". *Proceedings of 5th International Conference on Information-Warfare and Security*, 398–406. Academic Conferences Limited, England, 2010.

[26] Dube, T., R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers. "Malware Type Recognition and Cyber Situational Awareness". *Proceedings of the IEEE Second International Conference on Social Computing*, 938–943. IEEE, 2010.

[27] Dube, T., R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers. "Malware Target Recognition via Static Heuristics". *Journal of Computers & Security*, 2011. Submitted Apr. 12, 2011. Second revision.

[28] Duda, Richard, Peter Hart, and David Stork. *Pattern Classification*. Wiley, New York, NY, USA, 2nd edition, 2001.

[29] Edwards, C. and M. Riley. "Sony Data Breach Exposes Users to Years of Identity-Theft Risk", May 2011. URL www.bloomberg.com/news/2011-05-03/sony-breach-exposes-users-to-identity-theft-as-credit-card-threat-recedes.html. Accessed Jun. 22, 2011.

[30] Eilam, Eldad. *Reversing: Secrets of Reverse Engineering*. Wiley, Hoboken, NJ, USA, 2005.

[31] Endsley, M. "File:SA Wikipedia Figure 1 Shared SA (20Nov2007).jpg", Dec. 2007. URL en.wikipedia.org/wiki/File:SA_Wikipedia_Figure_1_Shared_SA_(20Nov2007).jpg. [Online; accessed 25-June-2011].

[32] Endsley, M. R. "Design and evaluation for situation awareness enhancement". *Proceedings of Human Factors Society 32nd Annual Meeting*, volume 1, 97–100. Human Factors Society, Santa Monica, CA, 1988.

[33] Endsley, M. R. "Toward a Theory of Situation Awareness in Dynamic Systems". *Human factors*, 37(1):32–64, 1995.

[34] Erdélyi, Gergely. "Hide 'n' Seek? Anatomy of Stealth Malware". *BlackHat Conference, Amsterdam, Netherlands, Europe*, volume 19. 2004.

[35] France-Presse, A. "Hackers Crack Pentagon F-35 Data", Apr. 2009. URL www.defensenews.com/story.php?i=4048737. Accessed Jun. 22, 2011.

[36] Girosi, F. and T. Poggio. "Representation Properties of Networks: Kolmogorov's Theorem Is Irrelevant". *Neural Computation*, 1(4):465–469, 1989.

[37] Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten. "The WEKA Data Mining Software: An Update". *SIGKDD Explorations*, 11.1, 2009.

[38] Hebb, Donald. *The Organization of Behavior*. Wiley, New Your, USA, 1949.

[39] Henchiri, O. and N. Japkowicz. “A Feature Selection and Evaluation Scheme for Computer Virus Detection”. *Proceedings of the Sixth International Conference on Data Mining*, 891–895. IEEE, 2006.

[40] Hörz, M. “HxD - Freeware Hex Editor and Disk Editor”, 2008. URL mh-nexus.de/en/hxd/. Accessed Oct. 19, 2008.

[41] Idika, Nwokedi and Aditya P. Mathur. “A Survey of Malware Detection Techniques”. *Technical Report, Department of Computer Science, Purdue University*, 2007.

[42] Kaspersky Lab. “Kaspersky Lab”, 2011. URL usa.kaspersky.com/. [Online; accessed 23-May-11].

[43] Keizer, G. “Is Stuxnet the ’best’ malware ever?”, Sep. 2010. URL www.computerworld.com/s/article/9185919. Accessed Jun. 22, 2011.

[44] Kephart, J. O. and B. Arnold. “Automatic Extraction of Computer Virus Signatures”. *Proceedings of the 4th Virus Bulletin International Conference*, 178–184. Virus Bulletin, Oxford, UK, 1994.

[45] Kivinen, J. and M. K. Warmuth. “Relative Loss Bounds for Multidimensional Regression Problems”. *Machine Learning*, 45(3):301–329, 2001.

[46] Kolter, J. Z. and M. A. Maloof. “Learning to Detect Malicious Executables in the Wild”. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 470–478. ACM, 2004.

[47] Kolter, J. Z. and M. A. Maloof. “Learning to Detect and Classify Malicious Executables in the Wild”. *Journal of Machine Learning Research*, 7:2721–2744, 2006. ISSN 1532-4435.

[48] Landis, J. R. and G. G. Koch. “The Measurement of Observer Agreement for Categorical Data”. *Biometrics*, 33(1):159–174, 1977.

[49] Lee, Tony and Jigar J Mody. “Behavioral Classification”. *Proceedings of EICAR*. 2006.

[50] Levenshtein, V. I. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. *Soviet Physics—Doklady*, 10(8):707–710, 1966.

[51] Virus Bulletin Ltd. “Windows XP Professional–April 2011”, Apr. 2011. URL www.virusbtn.com/vb100/archive/test?id=160. [Online; accessed 17-May-2011].

[52] Luke, James and C J Harris. “The Application of CMAC Based Intelligent Agents in the Detection of Previously Unseen Computer Viruses”. *Proceedings 1999 International Conference on Information Intelligence and Systems*, 662–6. 1999.

[53] Maloof, M. (editor). *Machine Learning and Data Mining for Computer Security*. Springer, London, UK, 2006.

[54] manpagez. "man page hexdump section 1", 2004. URL www.manpagez.com/man/1/hexdump/. Accessed Oct. 12, 2010.

[55] MathWorks, Inc. "Discriminant analysis", 2009. URL www.mathworks.com/help/toolbox/stats/classify.html. Accessed Dec. 14, 2009.

[56] MathWorks, Inc. "Polynomial eigenvalue problem", 2009. URL http://www.mathworks.com/help/techdoc/ref/polyeig.html. Accessed Jan. 6, 2010.

[57] MathWorks, Inc. "classregtree", 2010. URL www.mathworks.com/help/toolbox/stats/classregtree.html. Accessed Jan. 9, 2010.

[58] MathWorks, Inc. "Jarque-Bera test", 2010. URL www.mathworks.com/help/toolbox/stats/jbtest.html. Accessed May 13, 2010.

[59] MathWorks, Inc. "Lilliefors test", 2010. URL www.mathworks.com/help/toolbox/stats/lillietest.html. Accessed May 13, 2010.

[60] MathWorks, Inc. "MATLAB", 2010. URL www.mathworks.com.

[61] Merritt, David. *Spear Phishing Attack Detection*. Master's thesis, Air Force Institute of Technology, 2011.

[62] Microsoft Corp. "Microsoft Portable Executable and Common Object File Format Specification", 2008. URL www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx. Accessed Dec. 23, 2009.

[63] Microsoft Corp. "Microsoft Security Intelligence Report", 2009. URL www.microsoft.com/security/portal/Threat/SIR.aspx. Accessed Nov. 25, 2009.

[64] Mitre Corp. "Common Malware Enumeration (CME)", Apr. 2009. URL cme.mitre.org/index.html. Accessed Sept. 16, 2009.

[65] Moser, Andreas, Christopher Kruegel, and Engin Kirda. "Limits of Static Analysis for Malware Detection". *Annual Computer Security Applications Conference (ACSAC)*. 2007.

[66] Moser, E. "The Multivariate Normal Distribution", 2005. URL www.stat.lsu.edu/faculty/moser/exst7037/mvnprop.pdf. Accessed May 23, 2010.

[67] Nakayama, Takayoshi. "W32.Wargbot", Feb. 2007. URL www.symantec.com/business/security_response/writeup.jsp?docid=2006-081312-3302-99. Accessed Sept. 15, 2009.

[68] Norman ASA. "Norman Sandbox", Sept. 2009. URL www.norman.com/technology/norman_sandbox/en-us. Accessed Sept. 18, 2009.

[69] Okamoto, Takeshi and Yoshiteru Ishida. "A Distributed Approach to Computer Virus Detection and Neutralization by Autonomous and Heterogeneous Agents". *The Fourth International Symposium on Integration of Heterogeneous Systems*, 328–331. 1999.

[70] The WildList Organization. "The WildList Organization International", May 2011. URL www.wildlist.org. [Online; accessed 17-May-2011].

[71] Popa, Bogdan. "SourceForge.net Hacked!", Dec. 2007. URL news.softpedia.com/news/SourceForge-net-Hacked-73241.shtml. Accessed Mar. 3, 2010.

[72] Rafiq, Newaz and Yida Mao. "Improving Heuristics". *Virus Bulletin*, 9–12, August 2008.

[73] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". *Psychological Review*, 65(6):386–408, 1958.

[74] Russinovich, Mark. "Strings v2.41", Mar. 2009. URL technet.microsoft.com/en-us/sysinternals/bb897439.aspx. Accessed Mar. 18, 2009.

[75] Russinovich, Mark and Bryce Cogswell. "Process Monitor", Sept. 2009. URL technet.microsoft.com/en-us/sysinternals/bb896645.aspx. Accessed Sept. 18, 2009.

[76] Schipka, Maksym. "Heuristic detection of malicious code", Mar. 2009. URL patft.uspto.gov/. Accessed Mar. 15, 2009.

[77] Schultz, M., E. Eskin, E. Zadok, and S. Stolfo. "Data Mining Methods for Detection of New Malicious Executables". *IEEE Symposium on Security and Privacy*, 38–49. IEEE, 2001.

[78] Sciabica, Joe. "Industry Support to the Warfighter", Oct. 2008. URL www.afcea-infotech.org/2008/media/thursday/thu_pm1_sciabica.pdf. Accessed Nov. 13, 2009.

[79] ShadowServer. "Packer Statistics", Sept. 2009. URL www.shadowserver.org/wiki/pmwiki.php/Stats/PackerStatistics. Accessed Sept. 17, 2009.

[80] Siddiqui, Muazzam, Morgan Wang, and Joohan Lee. "A Survey of Data Mining Techniques for Malware Detection using File Features". *Proceedings of the 46th Annual ACM Southeast Regional Conference*, 509–510. ACM, New York, NY, USA, 2008.

[81] Siegel, Sidney and Jr. N. J. Castellan. *Analyzing Categorical Data*. Springer Verlag, 1988.

[82] Simonoff, J. *Analyzing Categorical Data*. Springer Verlag, 2003.

[83] Simonoff, J. "analcatdata.zip", 2008. URL lib.stat.cmu.edu/datasets/. Accessed Oct. 12, 2008.

[84] Sunbelt Software. "Sunbelt CWSandbox", Sept. 2009. URL www.sunbeltsoftware.com/Developer/Sunbelt-CWSandbox/. Accessed Sept. 18, 2009.

[85] Sung, A., J. Xu, P. Chavez, and S. Mukkamala. "Static Analyzer of Vicious Executables (SAVE)". *Proceedings of the 20th Annual Computer Security Applications Conference*, 326–334. IEEE, Los Alamitos, CA, USA, 2004.

[86] Swain, Bijay. "What are malware, viruses, Spyware, and cookies, and what differentiates them?", 2009. Accessed May 23, 2009.

[87] Symantec Corp. "Understanding Heuristics: Symantec's Bloodhound Technology". *Symantec White Paper Series*, XXXIV(1):1–14, 1997.

[88] Symantec Corp. "Scan Day for Virus Infections", 2011. URL www.symantec.com/about/news/release/article.jsp?prid=19960304_02. [Online; accessed 23-May-11].

[89] Symantec Corp. "Threat Explorer", 2011. URL us.norton.com/security_response/threatexplorer/azlisting.jsp. [Online; accessed 23-May-11].

[90] Szor, P. and E. Chien. "Code Red II", 2011. URL us.norton.com/security_response/writeup.jsp?docid=2001-080421-3353-99. [Online; accessed 23-May-11].

[91] Szor, Peter. *The Art of Computer Virus Research and Defense*. Addison-Wesley, Indianapolis, IN, USA, 2005.

[92] Tabish, S. M., M. Z. Shafiq, and M. Farooq. "Malware detection using statistical analysis of byte-level file content". *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 23–31. ACM, 2009.

[93] Tadjudin, S. and D. A. Landgrebe. "Covariance Estimation with Limited Training Samples". *IEEE Transactions on Geoscience and Remote Sensing*, 37:2113–2118, 1999.

[94] Tesauro, G. J., J. O. Kephart, and G. B. Sorkin. "Neural Networks for Computer Virus Recognition". *IEEE Expert*, 11(4):5–6, 1996.

[95] ThreatExpert Ltd. "Automated Threat Analysis", Feb. 2009. URL www.threatexpert.com/. Accessed Feb. 18, 2009.

[96] Treadwell, S. and M. Zhou. "A heuristic approach for detection of obfuscated malware". *Intelligence and Security Informatics*, 291–299. IEEE, 2009.

[97] Tsoumakas, G. and I. Katakis. "Multi-Label Classification: An Overview". *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[98] VirusTotal. "Free Online Virus and Malware Scan", Feb. 2009. URL www.virustotal.com/. Accessed Feb. 18, 2009.

[99] VX Heavens. "Virus Collection", 2010. URL vx.netlux.org/vl.php. Downloaded Apr. 15, 2010.

[100] Weber, Michael, Matthew Schmid, Michael Schatz, and David Geyer. "A Toolkit for Detecting and Analyzing Malicious Software". *Proceedings of the 18th Annual Computer Security Applications Conference*, 423. Citeseer, 2002.

[101] Xu, J. Y., A. H. Sung, P. Chavez, and S. Mukkamala. "Polymorphic Malicious Executable Scanner by API Sequence Analysis". *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, 383. IEEE, 2004.

[102] Ye, Y., D. Wang, T. Li, and D. Ye. "IMDS: Intelligent Malware Detection System". *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1043–1047. ACM, 2007.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)* 15–09–2011	**2. REPORT TYPE** Doctoral Dissertation	**3. DATES COVERED** *(From — To)* September 2008-September 2011

4. TITLE AND SUBTITLE A Novel Malware Target Recognition Architecture for Enhanced Cyberspace Situation Awareness	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Dube, Thomas E., Maj	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/DCE/ENG/11-07
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The rapid transition of critical business processes to computer networks potentially exposes organizations to digital theft or corruption by advanced competitors. One tool used for these tasks is malware, because it circumvents legitimate authentication mechanisms. Malware is an epidemic problem for organizations of all types. This research proposes and evaluates a novel Malware Target Recognition (MaTR) architecture for malware detection and identification of propagation methods and payloads to enhance situation awareness in tactical scenarios using non-instruction-based, static heuristic features. MaTR achieves a 99.92% detection accuracy on known malware with false positive and false negative rates of 8.73e-4 and 8.03e-4 respectively. MaTR outperforms leading static heuristic methods with a statistically significant 1% improvement in detection accuracy and 85% and 94% reductions in false positive and false negative rates respectively. Against a set of publicly unknown malware, MaTR detection accuracy is 98.56%, a 65% performance improvement over the combined effectiveness of three commercial antivirus products.

15. SUBJECT TERMS

Malware Target Recognition, Malware Detection, Malware Propagation, Malware Payload, Cyberspace Situation Awareness, Machine Learning

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Richard A. Raines
U	U	U	UU	180	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-6565, x4278; richard.raines@afit.edu