3-21-2013

Learning Enterprise Malware Triage from Automatic Dynamic Analysis

Jonathan S. Bristow

LEARNING ENTERPRISE MALWARE TRIAGE FROM AUTOMATIC DYNAMIC ANALYSIS

THESIS

Jonathan S. Bristow, Captain, USAF

AFIT-ENG-13-M-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

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

AFIT-ENG-13-M-10

LEARNING ENTERPRISE MALWARE TRIAGE FROM AUTOMATIC DYNAMIC ANALYSIS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Insitute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Jonathan S. Bristow, B.S.

Captain, USAF

March 2013

AFIT-ENG-13-M-10

LEARNING ENTERPRISE MALWARE TRIAGE FROM AUTOMATIC DYNAMIC ANALYSIS

Jonathan S. Bristow, B.S.
Captain, USAF

Approved:

Thomas E Dube
Maj Thomas E. Dube, PhD (Chairman)

28 FEB 13
Date

Rich A Raines
Richard A. Raines, PhD (Member)

28 Feb 13
Date

Rusty O. Bald
Rusty O. Baldwin, PhD (Member)

8-Mar-13
Date

COL Grant A. Jacoby, PhD (Member)

13 MAR '13
Date

AFIT-ENG-13-M-10

Abstract

Adversaries employ malware against victims of cyber espionage with the intent of gaining unauthorized access to information. To that end, malware authors intentionally attempt to evade defensive countermeasures based on static methods. This thesis analyzes a dynamic analysis methodology for malware triage that applies at the enterprise scale.

This study captures behavior reports from 64,987 samples of malware randomly selected from a large collection and 25,591 clean executable files from operating system install media. Function call information in sequences of behavior generate feature vectors from behavior reports from the files. The results of 64 experiment combinations indicate that using more informed behavior features yields better performing models with this data set. The decision tree classifier attained a max performance of 0.999 area under the ROC curve and 99.4% accuracy using argument information with function sequence lengths from 11–14.

This methodology contributes to strategic cyber situation awareness by fusion with fast malware detection methods, such as static analysis, to change the game of malware triage in favor of cyber defense. This method of triage reduces the number of false alarms from automatic analysis that allows a 97% workload reduction over using a static method alone.

Dedicated to our Lord, to my wife and our children, to my mother, to dear friends,
to liberty, and to the United States of America

Acknowledgments

Thank you, Jesus Christ, my Lord and Savior, for freely justifying me through faith in your atoning blood. You give me peace through seemingly trying times despite my myriad personal failures.

Thank you, my beautiful, loving bride. Your help exactly meets my needs, and you are a true blessing in my life. I also thank you for your sacrifices and overcoming in bringing our (currently) four children into this world, and for teaching and training them to walk in truth.

Thank you, mother, for always encouraging me academically. Your sacrifices have made this journey and these opportunities possible.

Thank you, Major Thomas Dube, PhD, my wise academic advisor. Your vision and advice inspires me. Thank you also to the members of my thesis committee. Your rigor has taught me a great deal about conducting research.

Also thank you to my fellow students and to the faculty and staff of the Air Force Institute of Technology.

Ultimately, thank you to the courageous people who have put their lives on the line in war and vigilance; striving for love of liberty so that we may enjoy the blessing of freedom from tyranny and usurpation in this blessed nation.

Jonathan S. Bristow

Table of Contents

Page

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
API	application programming interface
AUC	area under the receiver operating characteristic (ROC) curve
CFG	control flow graph
CUT	Component under Test
DLL	dynamic link library
ECI	enterprise cyber infrastructure
HCL	hierarchical clustering with complete linkage
kNN	k-nearest-neighbors
LSH	locality-sensitive hashing
MaTR	malware target recognition
MIST	malware instruction set
MLP	multilayer perceptron
NB	naïve Bayes
ROC	receiver operating characteristic
SUT	System under Test
SVM	support vector machine
WEKA	Wakaito Environment for Knowledge Acquisition

LEARNING ENTERPRISE MALWARE TRIAGE FROM AUTOMATIC DYNAMIC ANALYSIS

I. Introduction

Malware plagues enterprise networks. Malware authors intentionally attempt to evade defensive countermeasures. These adversaries employ malware against victims of cyber espionage with the intent of gaining unauthorized access to information or performing other malicious behavior such as corrupting data or denying access to information. As long as it is possible to use malware to achieve gain, then adversaries will attempt to introduce malware into enterprise cyber infrastructures (ECIs) [15].

Cyber defenders deploy a variety of responses to mitigate the threat of malware. To counteract *signature-based* malware detection, such as antivirus products and static analysis, malware authors implement a variety of obfuscation techniques that change the digital "appearance" of malware while preserving malicious behavior [9, 28, 31]. Initial results indicate that a *dynamic analysis* of malware can reveal malware hidden to static analysis by intentional obfuscation by observing the actual behavior of executable files [21, 26]. Furthermore, a dynamic analysis approach is significantly different from a static analysis approach such that one can refine the results of the other. That is, applying a dynamic method to the results of a previous static method can reduce the false alarms of both methods together for a more efficient malware detection system [6].

Manual review of large sets of dynamic analysis reports remains unfeasible because an enterprise network contains hundreds of thousands of unique executable files. *Malware triage* seeks to reduce the workload of the available cyber analysts by detecting the files that most closely resemble malware.

A given ECI contains *executable files* from a variety of sources. Operating systems install some executable files at install time or during updates, and others reside as part of an application that provides some set of features. Some applications directly support the requirements of the users of the ECI, while other applications provide a variety of security services, such as antivirus software, anti-spyware software or host-based intrusion detection system software. Another source of executable files on ECIs is administrative programs such as application installers or remote management software. Each application contains a set of executable files that interact together to provide the service of the application. Thus, each executable file behaves in a certain way, and there are a large variety of acceptable behaviors present from executable files on a typical ECI.

Additionally, some ECIs contain malicious executable files that provide a service to unauthorized users. Adversaries specifically program such executable files to perform malicious behaviors and covertly introduce them to the ECI. Examples include remote control of resources, information stealing or destructive actions. Cyber defenders must detect and remove these malicious executables in order to continue to meet the mission requirements of the enterprise.

This thesis examines the effects of feature extraction and selection on enterprise-level malware triage, and provides a methodology for behavioral analysis of unknown executable files with the goal of detecting malicious executables. Furthermore, this methodology contributes to strategic cyber situation awareness by combining with fast malware detection methods, such as static analysis, to change the game of malware triage in favor of cyber defense.

Analysis of the experiments validates using both application programming interface (API) argument information and behavior sequences of lengths from 11–14 to build more accurate executable classification models, and does not find a significant benefit of normalization. Given the high accuracy of 99.4% correct and low false positive rate of 1.75%, this method presents a prime candidate for a middle level dynamic method in a malware target recognition (MaTR) architecture [7].

Following is a summary of contributions of this thesis:

- critical analysis of recent automatic malware analysis research including a comparison to this study (Sections 2.3.2 and 2.4),
- analysis and discussion of results from 64 experiments on 3 key parameters of behavior analysis feature generation that the literature does not cover in a detailed manner (Chapter 4), and
- analysis of the contribution to a malware target recognition architecture by this dynamic analysis method, which establishes the feasibility of automatic behavior analysis at the enterprise scale (Sections 4.5 and 5.2).

This thesis covers a number of considerations for cyber defenders to prudently design a malware detection system, including the following summary. Cyber defenders using a malware detection system must:

- match the analysis sandbox environment to the enterprise environment,
- tailor the training set to the types of threats that face the particular enterprise,
- select features to represent the unique behaviors of benign and malicious programs,
- use an efficient dynamic analysis component,
- experiment with around eight levels of parameters when retraining,
- keep the number of training validation repetitions less than 10 (e.g. use 3 or 1),
- analyze the independence of the methods in different MaTR tiers, and
- add additional domain-specific information to the model (as possible).

II. Literature Review

This chapter introduces the background of machine learning (Section 2.1) and malware analysis (Section 2.2) before surveying recent research in dynamic analysis and automatic behavior classification (Section 2.3).

2.1 Machine Learning

The field of *Machine learning* involves using theories of statistics, algorithms, and knowledge representation to automatically represent information in a digital *model* of the real world. The *classification* process involves building a model on two or more distinct classes of training samples, and the model then attempts to predict the class of test samples. Decision trees build a classification model by repeatedly bisecting the input space based on a single attribute at a time. The tree building algorithm chooses the attribute that is most likely to evenly cut the space by measuring the information gain of all the available attributes according to the class labels. The support vector machine (SVM) algorithm finds a nonlinear classification boundary by selecting training samples that minimize the distance to the boundary [8]. The Wakaito Environment for Knowledge Acquisition (WEKA) platform provides implementations of many machine learning algorithms including J48, which implements the C4.5 decision tree algorithm [13, 24, 32].

Where classification uses training data that comes with class labels, *clustering* does not need to start with labels in order to put samples into groups. With clustering, a selected similarity measure (based on what makes sense for the data set) determines the relative distance between samples. Then the chosen algorithm dictates how the measurements shall determine which samples belong together in *clusters*. The hierarchical clustering with complete linkage (HCL) method finds the shortest distance between two existing clusters then combines the clusters by linking one sample from each cluster that maximizes the distance between those two samples. After repeating that algorithm until all the samples link to a single cluster a provided threshold cuts the resulting tree relationship into a set of clusters that each have a height that is no greater than the threshold [32].

Other learning algorithms, *weak learners*, perform only slightly better than random guessing (accuracy slightly above 50% correct for a two-class problem). *Ensembles* of weak learners combine many weak learners, each learning a different part of a problem to create a better-performing model [17]. Weak learners are fast to develop and execute, but some developers exert more research effort into developing a learning algorithm with more *heuristics*, which means they apply domain knowledge to solving part of the problem ahead of time. The resulting models perform better than weak learners, although more complicated learners necessarily have a higher computational cost [33].

In this effort, pilot studies show that decision trees perform better than bagging or boosting decision stumps and similarly to bagging or boosting decision trees. Ensembles of decision stumps train more quickly than decision trees, but decision trees train faster than ensembles of decision trees because each ensemble trains 10 models internally.

The practice of k-fold cross validation for building a robust classification model involves randomly splitting the training sample set into k equally-sized *folds* of samples. Then the learning algorithm builds a model with a training set of $(k - 1)$ folds, leaving one fold out. Then the algorithm uses the left-out fold as a testing set. Since the algorithm does not train the model with any samples from the test set during an iteration, the unknown samples validly measure the generality of the model. This process repeats k times, and each fold becomes the test set for one iteration. *Stratified* cross validation maintains class distributions throughout the method so that the relative size of the classes persists through different folds.

2.2 Static Analysis

A *static analysis* process reveals some attributes of an executable file without executing the code. As a result, the process quickly provides moderate detail. Egele et al. discuss various techniques for automatically analyzing malware and tools that implement such techniques [9]. Static analysis suffers from generic vulnerabilities to obfuscation by targeted malware. Moser et al. shows an approach for program transformation that defeats static analysis methods [21].

Eskandari and Hashemi combine a control flow graph (CFG) from disassembly information with an application programming interface (API) set to attain 97.77% accuracy on a set of 2,140 benign files from Microsoft Windows XP SP3 and 2,305 “network worms” from a repository at Shiraz University [10]. A CFG represents the possible actions that the program could take upon execution. They compare the 97.77% from CFG analysis to 92.19% accuracy using static *n*-grams as features. Both experiments use random forests on the same sample set. To get around the high processing time from the large graphs of CFG analysis, they flatten each graph into a feature vector using a sparse matrix representation. Disassembly-based information obtains fine-grained information, but it remains vulnerable to obfuscation. The paper does not report how much time the method takes for collection or analysis.

T.E. Dube attains 99.92% detection accuracy on a set of 31,193 samples of 32-bit malware from VX Heavens and 25,195 benign files from a clean install of Microsoft Windows from vendor media [7]. In comparison with the Kolter and Maloof *n*-gram method, Dube’s malware target recognition (MaTR) static method performs significantly more accurately at the 95% confidence level [16]. In his experiments, the best commercial antivirus product fails to achieve 50% accuracy on an unknown malware set. In addition, Dube’s static method averages less than one second of scan time for each file, whereas even the fastest antivirus product tested takes 43 seconds on average. Dube attains this result by deriving the feature set from proven static features using expert domain knowledge instead of the computationally heavy *n*-gram method.

Furthermore, Dube proposes a tiered architecture for cyber situation awareness [6]. In order to triage large amounts of unknown executable files, the bottom tier uses very fast methods that achieve a low false negative rate. When the bottom tier flags a sample as potentially malicious then the sample becomes an input to methods in the middle tier. The middle tier methods still have high detection rates, but also have low false positive rates, because the results from those methods go to cyber analysts at the top tier. Any false positives that reach the top tier are wasted overhead for the analysts, so the false positive rates of the underlying methods provide a way to directly measure the expected waste from overhead. This thesis (Section 5.2) provides insight into a behavior analysis method that fits into the middle tier of such an architecture to improve response times of cyber defenders by reducing the workload.

2.3 Dynamic Analysis

An appropriate *dynamic analysis* of an executable file reveals the most definitive information about its actions. Rossow et al. suggest some standard practices for malware experiment design such as removing benign programs from malware collections and commenting on the containment of the samples [27]. Bayer et al. introduce TTAnalyze (now Anubis, which also analyzes Android APK files), which uses Qemu emulation with Windows XP [4, 5]. TTAnalyze successfully reproduces and captures detailed data about the behavior of the executable file under analysis. TTAnalyze collects data at the level of the emulated processor, but it bridges the *semantic gap* with a kernel driver that leverages the `CR3` register and a userland process inside the guest. This means that the analysis method is able to obtain information about the state of the operating system to allow the process to correctly interpret the low-level instructions.

Yin et al. employ a whole system fine-grained dynamic taint analysis in Panorama [34]. Taint tracking labels the memory address to where a function returns a value, then records when another function uses that value as input or changes the value by writing into the same location again. This tracking allows a low-level emulator to gain insight into how programs interact with the operating system and produce a behavior report for the sandbox system. It collects information from an emulator on the entire guest system including high-level API calls and operating system interaction as well as an instruction trace with taint tracking. The system includes automated user actions during analysis such as typing text and browsing URLs. Upon testing with 42 malware samples and 56 benign samples, Panorama detects all the malware and only reports 3 false positives.

Egele et al. survey the literature on dynamic analysis techniques, tools, and analysis, but do not cover Cuckoo Sandbox, Windows 7 guests or 64-bit guests [9]. When a dynamic system analyzes an unknown executable file, the system may be able to choose which guest system is appropriate. If a malware author targets Windows 7, then the malware may not behave the same if the analysis system executes the file in Windows XP because of API differences between the versions. A similar phenomenon occurs with 64-bit malware on a 32-bit analysis system.

Moser et al. explores multiple execution paths during dynamic analysis by taking note of branching points and keeping track of the current state of execution [20]. Building on Anubis, Moser completes one iteration of execution, then reverts back to one of the branching points to continue analysis down a different path. This method can theoretically find behaviors of a sample that may not surface otherwise, such as behaviors that require user input or that wait for a specific time. However such completeness comes at the cost of computation time according the inherent *branching factor* of the program, which is how many alternate execution paths the program exhibits.

Lindorfer et al. attempt to *Disarm* malware that evades dynamic analysis [18]. Disarm works by submitting the samples to four different sandboxes and comparing the behavior reports. The findings indicate that several approaches produce a useful comparison for detecting anomalies [19]. Some malware authors include a capability to evade a certain analysis environment, but Lindorfer shows that most evading malware samples fail to evade in all environments under test. With a set of 1,686 samples, Disarm flags 431 (26%) as potentially evasive. Detailed analysis indeed finds timing attacks against Anubis to which a more plain Qemu sandbox is not vulnerable because it runs much faster. Other samples evade Anubis by exiting if execution starts with `explorer.exe` as the parent process. Lindorfer did note some false positives resulting from a peculiar (but not evasive) behavior by a certain family of malware.

2.3.1 API Call Sequences.

Trinius et al. introduce a *malware instruction set* (MIST), which is a *feature generation* technique that robustly represents a behavior action as a series of integers [29]. The API call name maps to an integer that represents a general category and another that uniquely represents that call name. The arguments also map to a hierarchical set of numbers that sequentially reveal more detailed information from left to right. Trinius also demonstrates *feature selection* over that representation by taking a level of numbers from the beginning of the malware instruction set (MIST) records as the training information while leaving out the rest of the data. As the level grows larger, the samples becomes more robust by including more detailed information. However, including too much specific information can reduce generality.

Rieck et al. takes a certain number of these segments in sequence to represent a chunk of behavior as a *q*-gram [26]. As in the Rieck paper, this thesis refers to sequences of behavior grams as *q*-grams, and uses the term *n*-grams to refer to bytes of binary data from the static method as in Kolter and Maloof. The MIST approach with behavior sequences allows researchers to tailor the level of analysis to the available computing resources. Using a larger *q*-gram representation exploits more details about the behavior of the executable file, but processing the data takes more space and time.

Each relevant API call belongs to a more general category. For instance, all the API calls that interact with the filesystem belong together, and all API calls that interact with the network interface belong together. Each category maps to an integer, and each specific call within each category maps to another set of integers. For example, the filesystem category is number `03` and the `MoveFile` API call is number `04` within that category. Therefore, the MIST report contains an entry `03 04` whenever a program moves a file. Sequences of instructions, *q-grams*, across the training set yield useful distance metrics between executable files for clustering and classification.

<table>
<tr><th>gram</th><th colspan="2"><code>09 02, 09 05</code></th></tr>
<tr><th>word</th><th>category</th><th>API</th></tr>
<tr><td><code>09 02</code></td><td>registry</td><td><code>OpenKey</code></td></tr>
<tr><td><code>09 05</code></td><td>registry</td><td><code>QueryValue</code></td></tr>
</table>

Figure 2.1: An example of a 2-gram at MIST level 1 with description of components

Figure 2.1 displays an example a 2-gram. The gram `09 02` refers to the *registry* API call `OpenKey`, and the gram `09 05` refers to the `QueryValue` API call which is also in the registry category. Hence, the 2-gram `09 02, 09 05` refers to the behavior of opening a registry key then querying a registry value. Without argument information, it is impossible to discern whether the executable is querying the value of the key that it just opened or if the query targets a different registry key.

03 05	00000001	00dc3932	00a93b39	002c392d	ba92d7c6
MoveFile	flags	source file ext	source file path	dest file file	dest file path

Figure 2.2: An example of a 1-gram at MIST level 2 with description of components

At MIST level 2, the grams include a level of function argument information. Figure 2.2 shows an example of a 1-gram at MIST level 2. First, the 03 05 part refers to the filesystem API MoveFile. Then there is a series of hash-encoded components that represent different parts of the argument information. MIST level 2 contains the more generic arguments, which would be common within a family of malware, but not arguments that are likely to be specific to a specific variant. For the MoveFile API, for example, the generic arguments include

- flags that represent filesystem move options,
- the source file, which includes

  the file extension and

  the path in the file system (not including the file name), and
- the destination file location, which also includes

  the file extension and

  the path.

On the other hand, MIST level 2 does not include the actual file base names. Such specifics would fall into a MIST level 3. Not every API call requires the same number of arguments, so only the arguments that are present in the behavior report get encoded into MIST format.

This representation also allows effectiveness of geometric clustering techniques, which Rieck et al. show performs efficiently [26]. When using prototypes for clustering, Rieck reports that a quad-core Opteron 2.4GHz system processes at a rate of 15,000 reports per day and uses 5GB of memory during regular clustering.

2.3.2 Behavior Analysis.

Rieck et al. implement hierarchical clustering with Euclidean distance complete linkage (HCL) [26]. Training starts with 3,133 samples from Sunbelt Software that come from 24 malware families that each have no more than 300 members. This labelled training set forms a reference to start the clustering process. After clustering on a set of 33,698 samples the algorithm finds 434 clusters which each contain 69 reports on average. Rieck shows high consistency of the top ten clusters with respect to Kaspersky labels, which indicates that the clusters represent the differing families of malware by behavior. Rieck explains that the majority of inconsistency that does occur comes from antivirus industry labels. The Rieck paper does not provide time measurements for collecting the dynamic analysis data.

Bailey et al. perform single-linkage hierarchical clustering on malware behavior [2]. They use a high-level view of behavior, recording only the *non-transient* changes to the system that persist after execution completes. For example, a malware file might enumerate the file system to get all the filenames present on the system then write those filenames to a file. Such behavior would be of value to an adversarial intelligence operative. Only the output file persists as evidence of the behavior of the malware, and the transient activity of the filesystem enumeration does not factor into their analysis. Bailey claims that this method avoids obfuscation of static analysis and low-level API sequences. They use the Backtracker system in VMware with Windows XP. They collect behavior data from 3,698 malware samples from the Arbor Malware Library (AML) over six months [1].

The $O(N^2)$ normalized compression distance step of the Bailey process takes the most time in both time and memory space as the number of samples rises to 500, compared to the preprocessing and clustering steps. The whole process takes about 220 seconds and only 300MB of memory for 526 samples. On a $3,698$ sample set, the method finds 403 clusters. Since 311 files do not exhibit any behavior during the process, Bailey claims a 91.6% detection rate, and then compares that detection rate to a 51.5% detection rate of Symantec. The report lists the common limitations of dynamic analysis, but does not assign a root cause to any of the files that failed to behave during observation. Bailey et al. do not run the process on non-malicious files to compare how closely other executable files compare to malicious files or to measure false positive tendencies of the high-level method [2].

Bayer et al. cluster 75,000 behavior reports within three hours and four gigabytes of memory with an Anubis system extended with *taint tracking* [3, 5]. As above, taint tracking labels the memory address to where a function returns a value, then records when another function uses that value as input or changes the value by writing into the same location again. This tracking allows a low-level emulator to gain insight into how programs interact with the operating system and produce a behavior report for the sandbox system. The blazing performance is due to the locality-sensitive hashing (LSH) clustering algorithm which approximates the distance measurements to achieve a good result quickly that is within a threshold parameter of the optimal solution.

Hu presents a malware detection system *MutantX* and a malware clustering system *Duet* [14]. The Duet dynamic analysis component uses binary features of n-grams of system calls from *strace* call traces (q-grams). This method employs the system call name and a canonical category to inform each datum, leaving out information from call arguments, which is similar to MIST level 1 [29]. The method does not specify how many features to select. Hu performs both static and dynamic analysis on 5,647 malware samples, and normalizes the feature vectors onto the unit circle. The static method computes n-grams of instruction sequences from a disassembly of the executable file. Hu notes that static analysis fails on 655 samples, while dynamic analysis fails on 645 samples. However, only with 72 samples do both methods fail. Comparing successful processing of samples using static 3- and 4-grams and dynamic 3- and 4-grams with combined behavior and static features, Hu finds 10%–15% improvement in successful processing with the combined information. This method fusion brings the clustering method to 98.72% coverage (72 samples failed of 5,647).

2.4 Summary

This thesis analyzes aspects of several other efforts. Table 2.1 summarizes similarities and differences of this and other works. None of those studies take advantage of as large a sample set, although Rieck is the closest with about a third as many, and none obtain samples from OpenMalware or US-CERT. Of the researchers that pursue a dynamic analysis approach, only Bailey does not capture API calls, instead noting only the persistent changes to the sandbox that remain following execution of the sample.

Two other of those efforts make classification between two or more sets a goal, while three seek to cluster a single body of samples by measuring similarity. One who choses to cluster implements an ensemble learning method, and one that classifies implements an ensemble (of a different sort). This study does not use ensemble methods because pilot tests show that fast ensembles are not as accurate as decision trees and accurate ensembles are slower than decision trees.

The details of these comparative studies reside in Table 2.2, where only one other uses a MIST representation for feature generation. Indeed, that research is first to publish the MIST, and while some other papers note the MIST in citations, none publish work that implements it. Only Kolter and Maloof use nearly as long gram structures, although that research uses static grams rather than behavior-based grams. Also, using long grams means the feature space gets very large, and only this study and Kolter and Maloof employ feature selection. Three papers mention normalizing feature vectors, but only this work publishes a comparison of normalized and non-normalized results.

Most of the studies use the same sorts of machine learning techniques. The k-nearest-neighbors (kNN) algorithm is popular both for clustering and classification, and the other clustering studies use HCL. The other two classification studies make a point to compare the common algorithms of naïve Bayes (NB) and SVM with the J48 decision tree in WEKA. Firdausi adds a multilayer perceptron (MLP) classifier to the model comparison, which trains in acceptable time with as few of samples in that study.

The set of studies that the summary tables cover is not exhaustive of all malware detection research, but the tables do contain the primary publications to date that bear major points in common with the research reported in this thesis.

Table 2.1: Overview summary of related work. An asterisk (*) denotes similarity to this research.

Author(s)	**analysis**	***N***	**source**	**features**	**learning**	**ensemble**
Bristow	*dynamic*	*90,578*	*OM/USCERT*	*API*	*classify*	*no*
Rieck [26]	dynamic*	33,698*	CWS	API*	cluster	no
Hu [14]	both	5,647	Symantec	instructions+API*	cluster	vote/bag
Bailey [2]	dynamic*	3,698	AML	persistent	cluster	no
Kolter [16]	static	3,622	MECS	*n*-grams	classify*	boosting
Firdausi [11]	dynamic*	470	Anubis	behavior;not specific	classify*	no

Table 2.2: Detailed summary of related work. An asterisk (*) denotes similarity to this research.

Author(s)	***l***	***n***	***M***	**norm**	**model**	**result**
Bristow	*1,2*	*1-16*	*500*	*both*	*J48*	*99.4% acc., 0.999 AUC*
Rieck [26]	1,2*	1-4	sparse/all	yes	HCL	80% recall
Hu [14]	1	3,4	all	yes	kNN	70% covg., 0.9 prec.
Bailey [2]	NA	1	all	no	HCL	91.6% acc.
Kolter [16]	NA	1-10*	10-10,000*	no	kNN,NB,SVM,J48*	0.9958 AUC
Firdausi [11]	1	1	116&11	no	kNN,NB,SVM,J48*,MLP	96.8% acc.

III. Methodology

Binary files execute a sequence of application programming interface (API) calls. This sequence represents the *behavior* of the executable file [26]. Let each API call along with the arguments represent one *action*. If a program writes to a file in the Windows operating system, the first action is to call the `CreateFile` API with arguments that identify the file to open for writing. On success, the API call returns a valid handle to the open file. With the handle, the program then calls the `WriteFile` API to put data into the file. A benign word processing program uses these API calls to save a users file, but some malicious programs use these API calls to save a record of keystrokes without user knowledge.

3.1 Problem Definition

The specific sequence of API calls defines the behavior of a program. Certain sequences occur in legitimate software, but to some extent different sequences occur in malware. This study examines the effects of malware instruction set (MIST) feature generation on enterprise-level malware triage.

3.1.1 Goals and Hypothesis.

The goal of this research is to determine an efficient and effective method to detect malware. The hypothesis is that certain feature selection parameter levels lead to machine learning performing with higher accuracy and efficiency at detecting malware compared to other levels.

This thesis addresses the following:

- Strategic Goal: Detect malware efficiently and effectively.
- Tactical Goal: Evaluate machine learning schemes for executable file classification.

- Hypothesis: Certain parameter levels enable more effective malicious file identification.

3.1.2 Approach.

The approach of this effort is to compare the efficiency and effectiveness of machine learning techniques with various levels of key feature selection parameters at classifying executable files. This study employs the MIST feature generation technique to encode behavior report information into a hierarchical format [26].

The experimental levels provide the basis for comparison relative to the same input sample set. The sample set results from random sampling of the large set. Standard techniques such as antivirus or previous analysis results validate the large sample set as malicious or not.

3.2 System Boundaries

The System under Test (SUT) in this experiment is a Malware Detection System (MDS). The MDS accepts a workload of known, labeled training sample executable files or unknown executable files, and it provides a malware detection service that identifies executable files which display malicious behavior. The dynamic analysis engine component creates dynamic analysis reports based on observed events. The feature generation component translates the behavior reports into MIST format and generates q-grams before selecting the most useful grams as features by filtering by information gain. The behavior analysis component is the Component under Test (CUT), which accepts sets of feature vectors and provides malware detection and classification capability. The block diagram in Figure 3.1 depicts the SUT and its components.

While this study measures the timing of a specific dynamic analysis engine known as Cuckoo Sandbox using VirtualBox, comparing timing measurements of different implementations is outside the scope of this study [12, 22]. The rest of the system

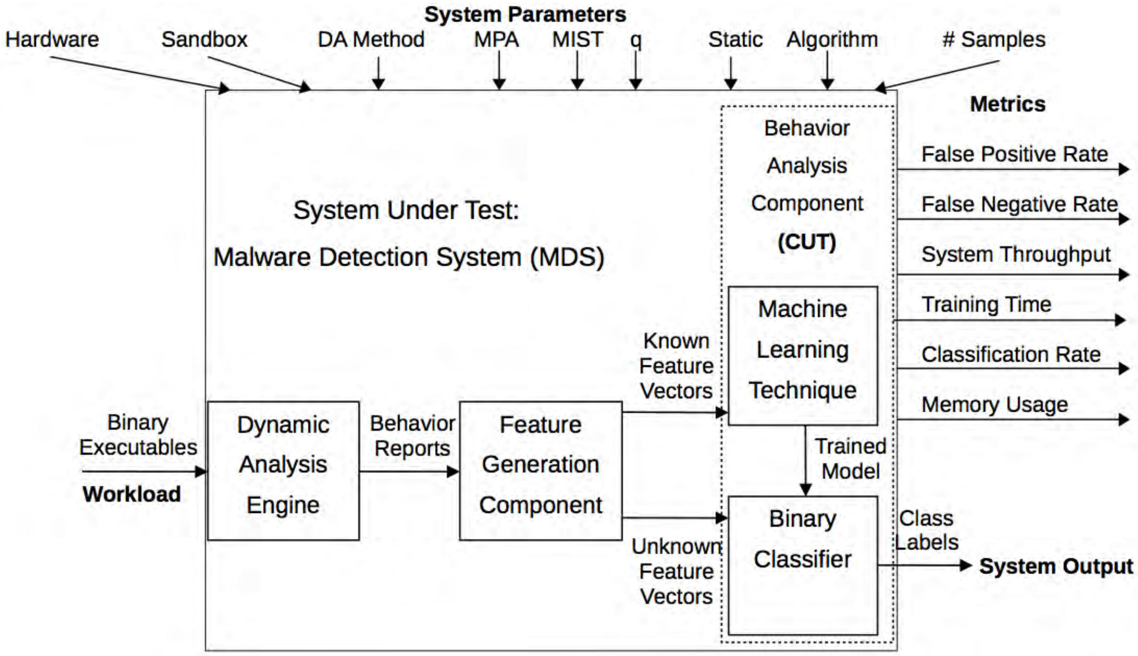

Figure 3.1: Malware Detection System Component Diagram

does not rely on a specific dynamic analysis engine implementation since all known implementations have the potential to create behavior reports that meet the requirements of the MIST feature generation technique. Similarly, this study does not depend on a specific computing platform. While this research effort validates this method on Microsoft Windows XP Service Pack 3 virtual guests, the concept extends to all other common operating systems where API call observation is possible through analogous methods.

3.3 System Services

The malware detection system detects malware within a set of unknown executable files by building a model from information the system discovers in a training set of executable files. A set of non-malicious *benign* files and a set of known malware samples comprise the training sample set. The system outputs a *cryptographic hash* of the executable file (for identification) along with the class label as the classifier determines.

The class labels derive from the labels correlated with the known training sample set. Thus, the approach is an example of *supervised learning*.

The success outcome is when the class label is correct for an executable file. A failure outcome is when the class label is incorrect. For failures, either the label indicates that the executable file is malicious when it is in fact benign, which is an example of a *false positive*, or the label indicates a benign file that is in fact malicious, an example of a *false negative*. False negatives are undesirable because they represent a missed opportunity to detect a malicious program, which means that an adversary retains the capability provided by that program. The false positive rate of a malware detection system allows operators to calculate how much wasted overhead the analysts must manually review.

The system discards samples if there is not enough behavior in a report and is therefore not desirable for input to clustering or classification algorithms. The cutoff threshold of the number of actions required is a result of applying domain knowledge and inspecting the smallest reports to find an appropriate level. There are 4,038 out of 90,578 total samples in this study that do not perform any actions. If a malware sample does not display behavior, then there is either some difference between the malware target environment and the test environment, or the sample simply does not perform any behavior. The system need not learn from nor detect malware samples that do not perform any behavior. Cyber defenders must take steps to ensure that a test environment matches the target environment in the enterprise in order to ensure that malware targeted for that enterprise performs behavior in the test environment.

Another failure outcome occurs when no features from the selection list of the top features, by information gain, that come from a sample. Such samples do not contribute information to that specific level of parameter levels. Thus a drawback exists from limiting the number of features. On the other hand, there exist millions of potential features at higher MIST levels and q-gram lengths, requiring feature vectors to reside in a space with hundreds of millions of dimensions. This *curse of dimensionality* requires attention when cyber analysts select a feature selection method for a machine learning scheme. This method deals with the large number of potential features by selecting the top 500 features according to information gain.

3.4 Workload

The workload this study provides to the SUT represents a set of executable files from an enterprise cyber infrastructure (ECI). A cryptographic hash of the contents of an executable file identifies the file in the workload. Identifying files by hash allows the system to treat two executable files that differ by a single bit or more as separate. Although file source data such as filename and source host are available to host analysis teams, such data are outside the scope of this experiment. A default installation of the Microsoft Windows XP Service Pack 3 operating system contains thousands of unique binary executable files, and a newly installed application may contain one to hundreds of executable files. Additionally, each update applied to an operating system or application adds or modifies one to hundreds of executable files at a time. Such updates introduce variability to a given ECI. Therefore, the distribution of input executable files to the SUT strongly depends on the individual ECI. The sample set consists of benign software samples similar to the most generic ECI. The distribution of specific software products does not necessarily affect the overall performance of a learned model because different versions of software that accomplish the same generic service likely exhibit similar behavior.

A current limitation of the Cuckoo Sandbox configuration pushes the ability to operate 64-bit guests outside the scope of this research, but the Cuckoo Sandbox developer intend to provide 64-bit capability in the future. Thus, this study uses 32-bit executable file samples in a 32-bit Windows XP SP3 guest. Cyber operators should ensure that the sample set for an operational malware detection system includes samples germane to the operational cyber infrastructure.

This study uses a set of malware from the US-CERT and Open Malware malware collections, which reflect many types of malware, including backdoors, constructors, sniffers, droppers, spyware, viruses, worms and trojans [25, 30]. This research randomly selects a subset of 64,987 32-bit Windows (Portable Executable format) malware samples from this large collection.

The *whiteware* set includes executable files from operating system vendor media, with the assumption that these executable files do not perform malicious behavior. Cyber defenders must add additional whiteware samples from installations of common types of user applications that occur in the enterprise cyber infrastructure. The white samples for this study originate from known clean Microsoft Windows media from Windows 2000 to Windows 7. This study uses a total number of 25,591 whiteware samples, and the total number of samples in the data set comes to 90,578.

The large number of malware samples from a very large collection precludes a detailed analysis of malware families within the scope of this study. However, this research uses a large number of samples randomly selected from a collection that contains a wide variety of malware. Therefore, the training set of malware executables has the potential to contain a wide variety of unique malicious behaviors. Other studies show that malware families usually perform similar behaviors, so whichever variants randomly appear in the training set contribute to the available training information for the learning algorithm. It is possible that a large malware family randomly present in the training set could introduce a bias toward detecting that family, however if such a family is more prevalent in the wild then detecting that family is a desirable trait. Operationally, cyber defenders should tailor the training set to the types of threats that face the particular enterprise.

3.5 Performance Metrics

The performance of the SUT comes from several measurements. The classification accuracy rate (%acc) of the malware detection model is the number of correctly-classified samples divided by the total sample size (percent correct). The false positive rate (FPR) is relevant for evaluating how many files the cyber analysts must manually review. The false negative rate (FNR) indicates the importance of a defense in depth strategy including alternative detection capabilities. A receiver operating characteristic (ROC) curve of a classifier graphs the true positive rate versus the false positive rate. Comparing several ROC curves shows the relative tradeoffs in false positives and false negatives. This study uses the area under the ROC curve (AUC), which is a summary of the ROC curve for a classifier, because it is simpler to compare 64 experimental classifiers by AUC than attempting to display and view all 64 ROC curve plots. Some relative false negative and false positive information is available in a full ROC plot that is not available in the AUC summary, but the AUC is suitable for this study.

In addition, the throughput of the SUT is the number of files that the system processes per unit time. The training time is the time the system requires to build a model from a specific machine learning technique with a given training set and feature selection parameter level.

3.6 System Parameters

Many parameters impact the performance of the SUT. The specific implementation of the dynamic analysis (DA) engine, feature generation component and machine learning component each require inspection of several relevant parameters.

3.6.1 Dynamic Analysis Engine.

Increased hardware capability increases the potential to execute additional jobs in parallel. This study utilizes available hardware to run 12 sandboxes in parallel. The operating system affects certain specifics of the implementation, but not the general concept under study [9]. This research uses a Dell server with two six-core Intel Xeon 2GHz processors and 500GB system memory. The operating system is 64-bit Ubuntu 12.04

Desktop (the sandbox environment requires the desktop version rather than the server version).

Emulation or virtualization has various effects, but the investigation of the differences of such effects is outside the scope of this study [9]. This study uses full operating system virtualization with Oracle VirtualBox [22].

Various types of hooking capture API calls; Cuckoo Sandbox uses *dynamic link library (DLL) injection*. The DLL injection method gets in the way of the test program calling API calls and logs all the calls before forwarding them to the operating system. Instruction-level tracing with data taint analysis captures behavior a different way, but an experimental comparison between the methods is outside the scope of this study. Any method that produces a MIST-compatible behavior report can contribute to this method.

Some publications do not report the dynamic analysis timeout, which is normally five minutes. This methodology employs a 15 second timeout in order to increase throughput. Comparing different timeouts is outside the scope of this study. The goal is to capture any malicious behavior during processing, but some files take a long time to execute. This study assumes that most malware completes malicious behavior quickly, within about five seconds. The timeout is higher, at 15 seconds, in order to allow sandbox initialization and the API hooking time to complete before the file executes. This assumption means that the system does not detect malware that waits 15 seconds or more to execute malicious behavior. However, even waiting for five minutes does not guarantee enough time to discover all malicious behaviors. A malware author is able to evade a detection system that has a particular timeout by finding out what the timeout is.

Multiple path analysis (MPA) might help solve the timeout problem. MPA increases the potential to detect obfuscation and avoid long delays, but requires additional computational time according to the branching factor of the file under analysis [20]. Experimenting with multiple path analysis is outside the scope of this study.

3.6.2 Feature Generation Component.

MIST level 1 records API name and category, and level 2 adds generic argument information (if present). Level 3 adds specific argument details. This study explores both level 1 and level 2. The parameter q is the length of q-gram instruction sequences. Previous studies use $q = 2$ or $q = 4$, and this study researches the effects of lengths from 1–16.

Feature selection uses information gain as a measurement of feature usefulness. The system keeps the top 500 features (q-grams), which is the same as the number Kolter and Maloof use as feature selection for training decision trees. Kolter and Maloof find 68,744,909 distinct static n-grams from a set of 476 malicious executables and 561 benign executables, and hence select the top 500 of those n-grams [16].

However, as Section 4.1 reports, this method finds from 85–4,171 grams at MIST level 1 and from 1,499,980–17,686,084 grams at level 2 although using a larger sample set of 90,578 samples total. This method finds far fewer distinct grams because the MIST behavior report gram space is more sparse than the binary file byte gram space in the static experiments. Since keeping 500 out of 68 million works best for the static n-gram method for Kolter and Maloof, then 500 should be sufficient out of 17 million features, since the features go to the same machine learning technique (J48). In addition, each feature from this behavior-based method potentially represents more information than arbitrary bytes extracted from the binary file. Therefore, this method should not require more features than the static n-gram method in order to represent useful information for the learning algorithm. However, if fewer features would perform just as well as 500, then including all 500 should only hinder computational burden and not classification accuracy. Therefore investigating the effects of different feature space sizes is outside the scope of this study.

3.6.3 Machine Learning.

This study uses the Wakaito Environment for Knowledge Acquisition (WEKA) J48 implementation of the C4.5 decision tree learning algorithm [13]. This study uses 64,987 malicious and 25,591 benign executable files, for a total set of 90,578 samples. The malicious files come from a combination of malware sets from US-CERT and Open Malware [25, 30]. The benign files come from known clean Microsoft Windows operating system install media. The experiments use *10-fold stratified cross validation* in order to measure generality. Each machine learning algorithm sees the same set of folds per repetition.

3.7 Factors

The factors for this experiment are the MIST level, the length of q-grams (q), and whether or not the feature vectors undergo normalization. This experiment evaluates two MIST levels: the level without any argument information and the level with partial, non-specific, argument information. The levels of q range from 1 through 16. Gram lengths longer than $q = 16$ lead to computationally prohibitive feature selection. The normalization factor includes two levels: non-normalized, which leaves the feature vectors as vectors of ones and zeros, and normalized, which applies basic vector normalization to project the magnitude of the vector onto the unit circle while maintaining the direction. Several authors mention these factors during similar research [2, 4, 16, 26].

Table 3.1: Factor Levels

Factor	**Levels**
MIST level (l)	1 – API call name only 2 – API name and generic arguments
Length of q-grams (q)	1–16
Normalization	Vector normalization (unit circle) Not normalized (binary 0/1)

3.8 Evaluation Technique

This experiment measures an instance of the CUT. Standard 10-fold cross-validation measures the generality of each model.

A number of the dynamic analysis runs undergo manual validation. This practice validates that the dynamic analysis component records malicious behavior.

3.9 Experimental Design

The methodology employs a full-factorial experimental design for a total of $2 \times 16 \times 2 = 64$ experiments. Each experiment undergoes 10 repetitions to explore the distribution variance in addition to the 10-fold cross validation, so each factor level undergoes 100 runs total. The cross validation is stratified so that class distributions remain similar throughout the process. Furthermore, each different experiment sees the same set of cross validation folds so that the relative mix of samples does not affect the variation in the results. Analysis uses a 99.9% confidence level to determine statistical significance. Since the goal for false negative rates is less than 0.1%, measurements need to have enough confidence to make a significant difference.

3.10 Methodology Summary

This method of malware detection involves detailed executable file classification. To determine which of the selected factor levels performs best in this domain, each factor tests on the same sample sets with the J48 learning algorithm. The input data are q-gram feature vectors from MIST feature generation based on dynamic analysis reports. Any dynamic analysis engine that can translate behavior reports into MIST format can compare to the results of this study.

IV. Results and Analysis

This chapter reports results and detailed analysis of the dynamic analysis engine, the feature generation component, and the machine learning component by the 6,400 experiment runs. First, Section 4.1 covers feature selection. Next, Section 4.2 presents the effects of the malware instruction set (MIST) level (l) and q-gram length (q) factors on classifier performance. Then Section 4.2.2 contains the effect of normalization, and Section 4.3 presents findings on the effective sample set size. Last, Section 4.6 analyzes limitations of this research and dynamic analysis at large.

4.1 Feature Generation and Selection

After dynamic analysis of the 90,578 samples, there are 4,038 samples that do not exhibit behavior. Table 4.1 displays the dynamic analysis results. Section 4.6 discusses reasons for those 4.46% of samples not yielding behavior. The 86,540 samples that do perform behavior, which make the other 95.54% of the total set, exhibit 85 different application programming interface (API) calls. That is, at MIST level 1, there are 85 unique 1-grams in the behavior reports. For example, in the 1-gram `03 04`, the number `03` refers to the filesystem category, and `04` refers to the `MoveFile` API call.

Table 4.1: Summary of dynamic analysis performance

Perform behavior	86,540	95.54%
Do not perform behavior	4,038	4.46%
Total	90,578	100.00%

Table 4.2 shows the relationship between the factor levels, the number of grams that q-gram analysis finds in the behavior reports, and the theoretical number of possible grams

Table 4.2: Detailed examples of feature generation component results

MIST	q	# grams	theoretical space size	% of space
1	1	85	120	70.83%
1	2	2,850	$85^2 = 7,225$	39.45%
1	3	20,665	$85^3 = 614,125$	3.36%
	⋮			⋮
2	1	1,499,980	N/A*	N/A*
2	2	3,722,381	$1,499,980^2 = 2.25 \times 10^{15}$	1.65×10^{-7}%
2	3	5,737,362	$1,499,980^3 = 3.37 \times 10^{18}$	1.70×10^{-12}%
	⋮			⋮

* There is no formal limit on unique argument data.

for three examples from each MIST level. When $q = 2$, there are 2,850 unique 2-gram sequences in the behavior reports, which is only 39.45% of the possible 2-long sequences of those 85 API calls. For example, the gram `09 02` refers to the registry API call `OpenKey`, and the gram `09 05` refers to the `QueryValue` API call which is also in the registry category. Hence, the 2-gram `09 02, 09 05` refers to the behavior of opening a registry key then querying a registry value. Without argument information, it is impossible to discern whether the executable is querying the value of the key that it just opened or if the query targets a different registry key.

As the value of q increases, the number of unique q-grams that occur in the behavior reports also increases, but not as fast as the number of possible grams. Each additional entry in a sequence multiplies the total possible number of permutations of grams by the number of possibilities for that entry (e.g. 85 in this data set). Hence, 3-grams have $85^3 = 614,125$ possible unique values, and 4-grams have $85^4 = 52,200,625$ possibilities. Thus the space of possible grams increases exponentially with gram length, but the grams that occur in the data set do not fill up that space.

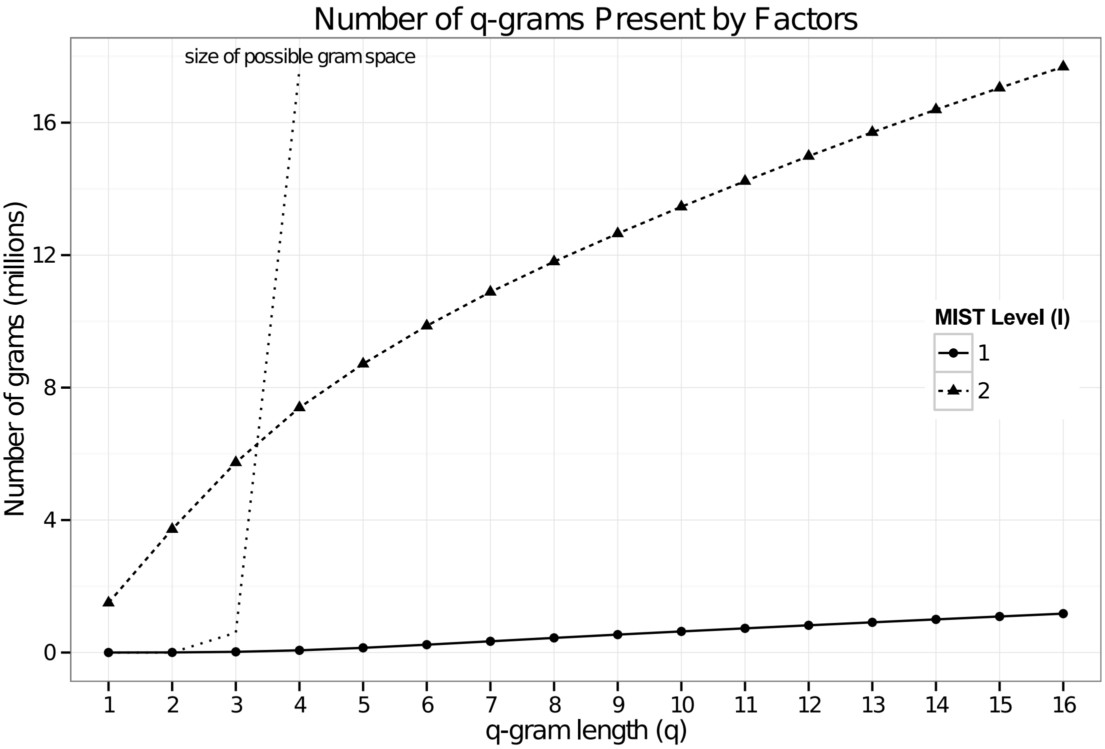

Figure 4.1: Graph of the number of millions of unique q-grams present in the data for each level of the MIST level (l) and gram length (q) factors with depiction of the size of the space of possible grams for MIST level 1

For MIST level 2, which records one level of function arguments, the number of possible gram variations is much higher. There are 1,499,980 unique 1-grams within the behavior reports of this data set at MIST level 2. This occurs because the arguments can be any value that the program could provide to that API call. Figure 4.1 indicates that, as with MIST level 1, the number of unique q-grams does not increase exponentially, but rather appears to increase linearly (or logarithmically) as the value of q increases.

4.2 Classifier Performance

This section reports that the MIST level factor contributes the largest effect to classifier performance and normalization does not significantly affect performance with this data set. The level of significance for the confidence intervals is 0.001 (i.e. 99.9% confidence that the true mean falls within the interval).

4.2.1 Overview.

Figure 4.2 shows that MIST level 2 dominates level 1 on this data set. Over the entire data set, MIST level 2 averaged an area under the receiver operating characteristic (ROC) curve (AUC) of 0.99106 ± 0.00065 at the 99.9% confidence level. The additional information that MIST level 2 includes over level 1 appears to better inform the resulting decision tree model. Additionally, 10 repetitions are clearly sufficient to characterize the majority of the variation amongst runs of the algorithm on this data set. This could mean that a lower number of repetitions would still prove sufficient in an operational environment where saving computation time improves reaction time. The following subsections provide more detailed analysis regarding the results of the experiments.

4.2.2 Normalization.

Figure 4.2 also shows that normalization does not attain a statistically significant effect on classifier accuracy at the 99.9% confidence level (nor at 95% confidence). Machine learning methods usually use normalization to reduce the bias of samples that contain a larger proportion of features because normalizing sets the magnitude of each feature vector to one without changing the direction of the vector.

Since normalization does not affect classification performance with this data set, then either the classifier is not sensitive to sample vectors that have a comparatively large magnitude, or the data set does not contain very many samples that yield large vectors. A larger feature vector is the result of a sample that performs more behaviors that the feature selection filter accepts. A data set does not fully demonstrate the normalization benefit if

MIST level	AUC	99.9% C.I.
1	0.9823	±0.00045
2	0.9911	±0.00065

Normalization	AUC	99.9% C.I.
non-norm	0.9866	±0.00068
norm	0.9868	±0.00055

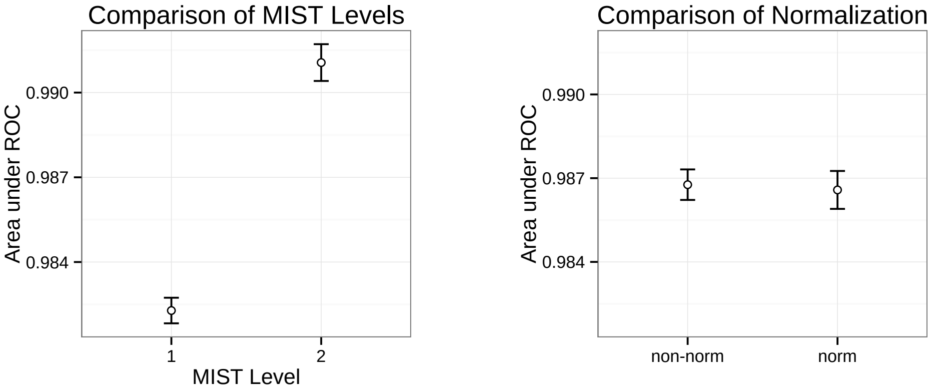

Figure 4.2: Details and chart of area under ROC comparisons of classifier performance by MIST level and normalization with 99.9% confidence intervals

there is a very small percentage of vectors with more attributes present compared to other vectors.

4.2.3 MIST and q Details.

Figure 4.3 shows that MIST level 2 consistently performs at a higher AUC than MIST level 1 except where the q-gram length reaches 15 and 16. MIST level 1 also reaches lows at $q = \{15,\ 16\}$. The 99.9% confidence intervals validate that the differences between the means are significant for the rest of the levels. Some potentially outlying data points include $q = \{7,\ 15,\ 16\}$ for both MIST levels 1 and 2 because each of those points are greatly lower than the points around them. Further examination of those outlying results continues below in Section 4.3. Appendix B provides tables with further details on measurements from each experiment.

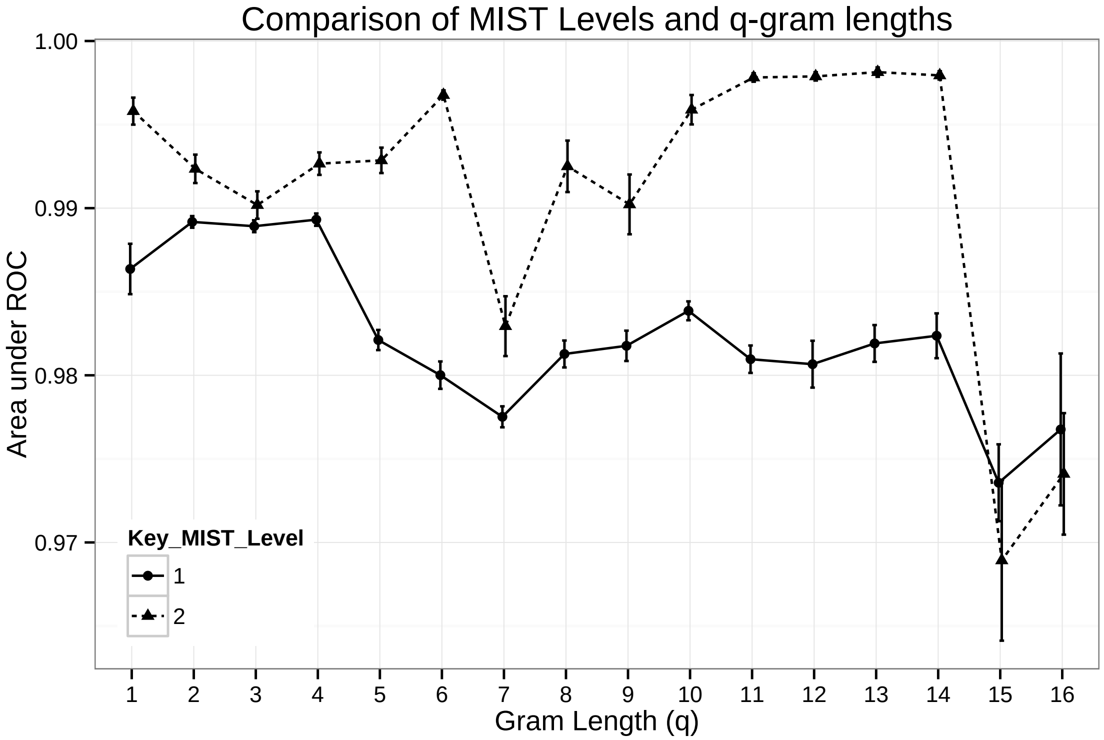

Figure 4.3: Area under ROC comparison of classifier performance by MIST level and *q*-gram length with 99.9% confidence intervals

4.3 Number of Samples

Of 90,578 samples total, 86,540 samples, 95.54%, yield behavior in this dynamic methodology. Figure 4.4 shows that the number of samples steadily decreases as the levels of the factors increases, which reveals that a large percentage of samples from the dynamic analysis results lose representation at high MIST level and *q*-gram length.

There are more unique *q*-grams both when grams are longer and when adding argument data. Thus, the 500 features that win selection comprises a much smaller percentage of the set of all possible features. Hence many samples no longer contain a selected behavior. This effect seems to be especially strong for $q = \{15, 16\}$.

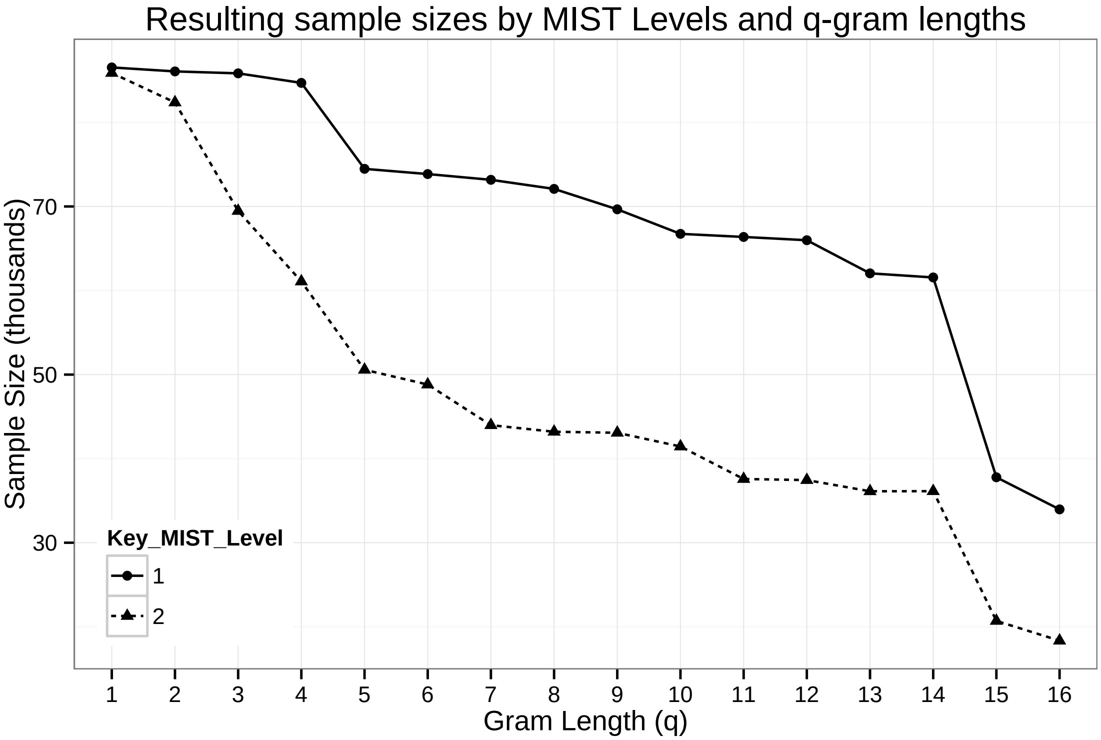

Figure 4.4: Comparison of resulting sample sizes by MIST level and gram length

The size of the sample set does not affect model performance as much as the amount of information available from each sample. Figure 4.5 at first shows a slight positive correlation between sample set size and classification accuracy for MIST level 1, however there seems to be a compounding factor. The labels of the interesting points from Figure 4.3 indicate that the lowest performance coincides with the fewest available samples when $q = \{15,\ 16\}$. Ignoring those outlying cases seems to reveal a slight positive correlation for MIST level 1, but a slight negative correlation with MIST level 2. Therefore, a low sample size adequately explains why learning performance is comparatively lower for $q = \{15,\ 16\}$. However, sample size does not explain the low performance at $q = 7$, which is especially pronounced in MIST level 2. On the other hand, false positives and false negatives are not out of the ordinary at $q = 7$, even though those values do change wildly with $q = \{15,\ 16\}$. The same is true for classification accuracy by the percent correct measure.

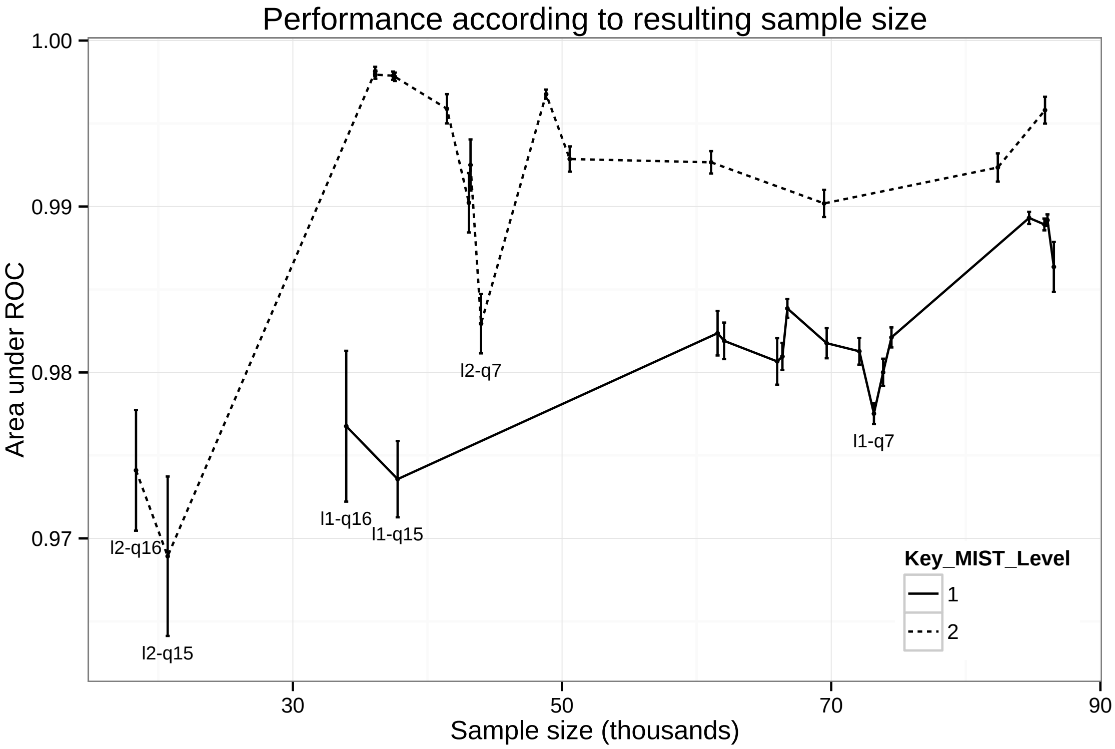

Figure 4.5: Performance with 99.9% confidence intervals according to sample set size after removing zero-vectors with selected labels by MIST level (l) and q-gram length (q)

The increasing levels of the MIST and q-gram factors yield feature vector sets of decreasing sizes because the methodology discards vectors that equal zero. Such vectors do not provide useful information to a model because it represents an executable file that exhibits no behavior. However, some files that do exhibit behavior end up with a feature vector of zero because only 500 features survive feature selection. This feature size parameter agrees with the number that Kolter and Maloof find useful for learning based on static 4-grams [16]. The usefulness of this level of the parameter arises from both the learning algorithm and the implicit dimensionality of the input data. Although this methodology differs from Kolter and Maloof regarding the source of the input data (the implicit dimensionality may differ), the learning algorithm is the same. That is, since 500 features works for the decision tree algorithm of Kolter and Maloof, then it is feasible that 500 features approximates a useful feature size for the decision trees in this study.

Kolter and Maloof find 68,744,909 distinct static *n*-grams from a set of 1037 samples total, and hence select the top 500 of those *n*-grams [16]. However, this method finds only 17,686,084 behavior *q*-grams using a sample set of 90,578 samples total. This method finds far fewer distinct grams because the MIST behavior report gram space is more sparse than the binary file byte gram space in the static experiments. Table 4.3 shows the selection rates for the first three levels of *q* for MIST levels 1 and 2.

Table 4.3: Detailed examples of feature selection rates

MIST	*q*	# grams	features	% of grams
1	1	85	85	100.000%
1	2	2,850	500	17.544%
1	3	20,665	500	2.420%
	⋮			⋮
2	1	1,499,980	500	0.0333%
2	2	3,722,381	500	0.0134%
2	3	5,737,362	500	0.0087%
	⋮			⋮

Since keeping 500 out of 68 million works best for the static *n*-gram method for Kolter and Maloof, then 500 could be sufficient out of 17 million features, since the features go to the same J48 machine learning technique. This means that very similar feature vectors input to machine learning in both studies. In addition, each feature from this behavior-based method potentially represents more information than arbitrary bytes extracted from the binary file. Therefore, this method represents more semantic information per feature than the static *n*-gram method.

Operationally, cyber operators should select a set of features that is large enough to represent the number of unique behaviors that benign and malicious programs perform. Then the machine learning algorithm discovers the relationship between the behaviors and the maliciousness of executable files from the training samples.

4.4 Timing Analysis

The dynamic analysis component takes 12 days to generate behavior reports from the 90,578 samples by running 12 parallel guests in Sub VirtualBox with Cuckoo Sandbox on a Dell server with two six-core Intel Xeon 2GHz processors and 500GB system memory on Ubuntu 12.04. The analysis timeout is 15 seconds, but each sample experiences an additional 54 seconds of overhead on average. The overhead mainly results from processing large behavior report files from executables that log a large number of API calls. Optimizing the dynamic analysis process for speed is outside the scope of this study because commercial dynamic analysis products solve this problem.

Translating a generic behavior report out of a dynamic analysis engine into the MIST format takes less than a second for small reports, and operates in time proportional to the length of the behavior report. Extracting the q-grams out of the MIST reports takes 2.4 minutes at MIST level 1 and $q = 1$, and it takes 6.9 hours for MIST level 2 with $q = 16$. Naturally, this processing time is proportional to the number and size of grams.

Training decision tree models with the Wakaito Environment for Knowledge Acquisition (WEKA) J48 implementation of the C4.5 algorithm on the 6400 experiment runs takes 81 days worth of computational time [23]. Using 20 parallel processes on the same hardware as above takes 4 days in the WEKA experimenter [13].

The dynamic analysis component and the experiments together take 16 days to run on these 90,578 samples. In an operational environment where time matters, a commercial dynamic analysis component could process 50,000 samples or more per day depending on the available level of investment in the enterprise for definitive malware detection. Furthermore, if only eight levels of parameters need exploration, and the number of repetitions can be less than 10, then training a detection model can take under one hour.

After building the model, each suspicious sample undergoes dynamic analysis, which takes 79 seconds on average during this study, then the resulting behavior report uses the same MIST global settings as the training set for translating into the MIST format. Then q-gram extraction uses the same indices as the training set to create a feature vector for the sample, which provides an input to the detection model. A decision tree classifies input in logarithmic time relative to the size of the tree, which averages 0.043 seconds of computational time during the experiments in this study.

Therefore, if an enterprise accomplishes this analysis method with a baseline of their approved software along with a set of malware that includes current threats, then they can have a functioning detection model in two business days. Then the dynamic analysis component and the model can classify suspicious samples at a rate of 35 samples per hour (again assuming commercial processing of 50,000 per day).

4.5 Operational Analysis

The information that a machine learning algorithm encodes into a model intrinsically comes from the input data. Therefore, the degree to which a model correctly reflects the real world depends on how representatively the input data captures relevant information about the real world goals. In a malware behavior model, an ideal data set contains samples that exhibit all the malicious behaviors to which an enterprise cyber infrastructure (ECI) is vulnerable. On the other hand, the machine learning algorithm consumes computational resources for each instance in the input data, so the ideal sample size for input data is the number of samples that completely contains the necessary malicious behavior sets without any duplicates of those behavior sets. However, fully identifying such a set requires complete behavior analysis of the entire available set, including all unknown malware behaviors from the future of a given ECI. Such a requirement takes the computational cost of obtaining the training set from high to impossible. On the other hand, a nearly complete set maintains usefulness so long as it contains sufficient behavior sets to identify some of the malware present in an ECI, which allows cyber operators to target different techniques capable of discovering additional malware.

The top 10 false positive rates of the data sets appear in Table 4.4. The lowest average false positive rate is 1.75%, achieved by MIST level 2 with $q = 4$. The top-performing data sets $q = \{13, 14\}$ appear next, followed by some shorter grams. All 10 come from MIST level 2.

Considering an example operation on an enclave network, suppose there are 10,000 executable files on hosts in the enclave. A team of eight cyber analysts deploy to the enclave in response to indications of an intrusion, and indeed there are 100 malware files hidden somewhere on the enclave. If each of the 8 cyber analysts can process 25 files per day, then to manually inspect 10,100 files it would normally take $10{,}100/25/8 = 50.5$ days to find the malware. Assuming that the average accuracy of 99.4% and false positive rate of 2.2% of the MIST level 2 and $q = 14$ model applies to the example operational file set, this method would falsely classify $10{,}000 \times 0.022 = 220$ benign files as malicious and detect $100 \times 0.994 = 99$ malicious files. The team of analysts can now process the 319 files manually and find all but one of the malware in $319/25/8 = 1.6$ days, which is over a 97% workload reduction.

Expanding this concept to the next level, consider a fast malware detection method, such as the static component of the Dube malware target recognition (MaTR) architecture. Dube's static method reaches 99.9% detection with a 0.1% false positive rate [7]. Given

Table 4.4: Top 10 Data Sets by False Positive Rate (FPR) at the 99.9% confidence interval (C.I.)

MIST	q	N	FPR	std. dev.	std. err.	99.9% C.I.
2	4	200	0.0175	0.003857	0.000273	± 0.0009
2	14	200	0.0220	0.006167	0.000436	± 0.0015
2	13	200	0.0239	0.007015	0.000496	± 0.0017
2	2	200	0.0264	0.005332	0.000377	± 0.0013
2	1	200	0.0283	0.004529	0.000320	± 0.0011
2	3	200	0.0302	0.006037	0.000427	± 0.0014
2	5	200	0.0327	0.008630	0.000610	± 0.0020
2	6	200	0.0338	0.006561	0.000464	± 0.0015
2	10	200	0.0372	0.008177	0.000578	± 0.0019
2	12	200	0.0383	0.008639	0.000611	± 0.0020

these rates, the same team of eight cyber analysts as in the previous example in two days could find the malware from the intrusion in a set of 10,000/0.001 = 10,000,000 files.

The operational result above assumes that the malware detection methods operate independently of each other. That is, the methods do not use the same information from the data set in order to make predictions. Dube's MaTR static method relies on a set of static features, such as structure anomalies, but a dynamic method like the one analyzed in this thesis relies on the actual observed behavior of the files. Those information sets arise from independent analysis methods. Figure 4.6 shows an example of two independent classifiers attempting to detect malware from an intrusion on a cyber infrastructure. The outside box represents all the executable files on the network, and the circles for classifiers *A* and *B* show the files that each is able to classify. The dotted line depicts a decision boundary, and the wavy lines show what each model classifies as malware. The labels show the relationships between false positives and false negatives. A false positive occurs when a model classifies a benign sample as malicious, which means that a higher level must spend extra resources validating that sample. The false positive rate of the MaTR architecture reveals a measurement of wasted overhead by the malware analysts. A false negative occurs when a model classifies a malicious sample as benign, which means the adversary retains the capability of that malicious executable.

The goal for organizing malware triage is to get all the malware within the wavy lines while pulling the overlap of the classifiers *A* and *B* apart as much as possible to reduce false positives. In practice, most classifiers do not leave many samples unclassified, so the circles would expand to fill in most of the box.

One exception to independence could be if the static method uses embedded dynamic link library (DLL) import strings as features, which relates to the possible API calls that the file could call. However, this behavior analysis feature generation component includes argument information that is not as available to a static method. In addition, the feature

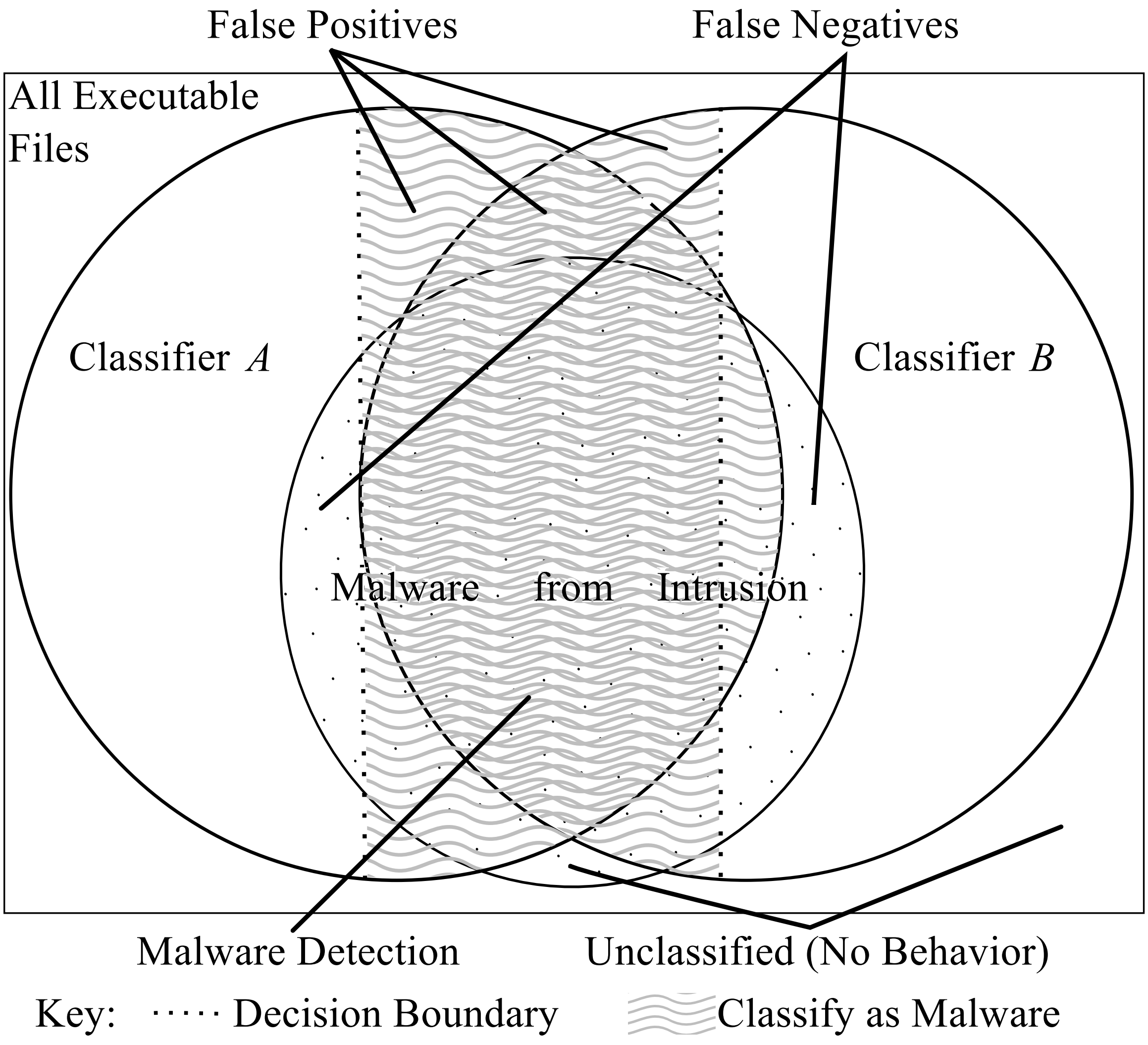

Figure 4.6: Venn diagram of independent malware classifiers

generation component considers a sequence of API calls, which would likely not be in the same order as listed in the imports. Therefore the main limitation on a powerful synergy between two independent methods in such an architecture is how closely the models perform with an operational data set rather than the more widely available data sets that academic studies use.

4.6 Limitations Analysis

The malware detection system has some limitations. There exist limitations in the malware set workload, the dynamic analysis method, and feature extraction and selection.

4.6.1 Malware Set Limitations.

This malware set comes from USCERT and Open Malware [25, 30]. The set contains a wide variety of types of malware, such as worms, downloaders and backdoors, and the set contains a wide variety of families within each type. Because the set contains a large number of malware samples, the members collectively exhibit a large number of malicious behaviors. The fact that the data only exhibit 85 of the possible 120 API calls raises questions of completeness. While some API calls are not often useful to routine types of programs, future research should validate that the training set covers enough different behaviors to be fully useful for training malware detection models. Other studies show that malware families usually perform similar behaviors, so whichever variants randomly appear in the training set contribute to the available training information for the learning algorithm. It is possible that a large malware family randomly present in the training set could introduce a bias toward detecting that family, however if such a family is more prevalent in the wild then detecting that family is a desirable trait. Operationally, cyber defenders should tailor the training set to the types of threats that face the particular enterprise.

4.6.2 Dynamic Analysis Limitations.

Not all executable files exhibit behavior during dynamic analysis. Several reasons contribute to this limitation, and this method discards all such samples. First, the operating system does not execute corrupted files that break operating system conventions. Despite the malware author's intentions, no malicious behavior executes in this case.

Second, some executable files require provision of command-line arguments, the presence of an encryption key, or to read from a configuration file or a DLL. Incident response possesses the capability to discover such items and input them into the dynamic analysis process for newer samples. Then the dynamic analysis process can call malicious functions of a DLL file. These examples demonstrate the importance of additional domain-specific information to improve malware detection operationally.

Some malware samples attempt to evade analysis by failing to exhibit malicious behavior. Such samples evade the method in this study. However, in a growing trend, many ECIs include virtualized servers and workstations where such samples also fail to perform malicious behavior. Other malware samples evade analysis by targeting the dynamic methodology. Lindorfer et al. suggest a mitigation by sending suspicious samples through a gauntlet of methodologies [18]. There remains the case where malware authors specifically design an executable file to evade such a defense (e.g. when adversaries have specific intelligence), and cyber operators must detect this intrusion from a different perspective (e.g. network traffic anomaly analysis). Moser et al. presents an additional mitigation with multi-path analysis [20]. This technique exploits instruction trace dynamic analysis by following both possible paths at every branch. Multi-path analysis suffers from performance concerns, but exposes more behaviors of an executable file.

V. Conclusions

This chapter summarizes the research conclusions and significance. Section 5.1 starts with a summary of conclusions on the research followed by the significance of the research in Section 5.2. Finally, Section 5.3 offers suggestions for future work on related subjects. Following this chapter are appendices that provide an additional level of detail of experimental results. Appendix A shows some of the top results sorted by different measurements, and Appendix B provides a summary of the results of all 64 experiments.

5.1 Research Conclusions

This research provides insight into the effect of malware instruction set (MIST) level, q-gram length, and normalization factors on feature selection of behavior reports for machine learning performance. Including generic argument information with MIST level 2 improves the classification accuracy of a decision tree learner on this data set. Additionally, the performance of the model should increase operationally with additional benign samples from the enterprise environment. The normalization factor is not significant with this data set, and would only significantly improve classification of data sets that include samples which exhibit a wider variety of different behaviors.

The size of samples available for model training after dropping zero-vectors depends on the number of unique grams which gram analysis discovers. Therefore, cyber defenders employing a similar method must include an appropriate number of features for the data so that the feature vectors represent more samples. Increasing the number of features also increases the machine learning computational requirements, so operators must choose a scheme that performs adequately on the available computing resources. This thesis covers a number of considerations for cyber defenders to prudently design a malware detection system, including the following summary. Cyber defenders using a malware detection system must:

- match the analysis sandbox environment to the enterprise environment as closely as possible,
- tailor the training set to the types of threats that face the particular enterprise,
- select a set of features that is large enough to represent the number of unique behaviors that benign and malicious programs perform,
- use an efficient dynamic analysis component depending on the available level of investment,
- experiment with around eight levels of parameters when retraining,
- keep the number of training validation repetitions less than 10 (e.g. use 3 or 1),
- analyze the independence of the methods in different malware target recognition (MaTR) tiers, and
- add additional domain-specific information to the model (as possible).

These suggestions are not exhaustive, but they follow from the experiments of this thesis. Of course, cyber operators should keep up to date on other research in the field to continue to proactively adapt to changing threats.

5.2 Research Significance

This research contributes to the body of knowledge available to researchers studying malware detection and operators employing a malware detection system. This study contributes valuable insight to a hierarchical cyber defense organization, such as Dube's MaTR, where a three-tiered malware triage structure funnels a large number of unknown executable files up to the human analyst tier [7]. The bottom tier consists of fast malware detection methods, such as static analysis methods. The bottom tier must not permit a high false negative rate, but it may admit a moderate false positive rate because the middle tier should reduce false positives.

The middle tier includes slower methods, such as dynamic analysis, which require time to emulate or simulate execution. The role of the middle tier is to decrease the number of false positives to send to the top tier, so the detection methods must be independent to those in the bottom tier. If the methods in the middle are not independent, then they would produce nearly the same results on the same input. However, an independent method produces different results such that the predictions of both methods cover the same malware (according to accuracy), but cover different false positives. Automatic behavior analysis is independent to static *n*-gram analysis because behavior analysis directly detects the malicious behavior rather than arbitrary byte strings.

Following is a summary of contributions of this thesis:

- critical analysis of recent automatic malware analysis research including a comparison to this study (Sections 2.3.2 and 2.4),
- analysis and discussion of results from 64 experiments on 3 key parameters of behavior analysis feature generation that the literature does not cover in a detailed manner (Chapter 4), and
- analysis of the contribution to a malware target recognition architecture by this dynamic analysis method, which establishes the feasibility of automatic behavior analysis at the enterprise scale (Sections 4.5 and 5.2).

As Section 4.4 discusses, if an enterprise accomplishes this analysis method in addition to a method with similar performance to Dube's static method with a baseline of their approved software along with a set of malware that includes current threats, then they can have functioning static and dynamic detection models within two business days after constructing the system and gathering the training samples. Such a malware target recognition (MaTR) can change the game of malware triage in favor of cyber defenders at enterprise cyber infrastructures (ECIs) in government, business and academia.

5.3 Future Work Recommendations

The following subsections describe considerations for future malware analysis researchers who investigate automatic behavior analysis starting with data set issues in Section 5.3.1. Next, Section 5.3.2 discusses additional machine learning algorithms, and lastly Section 5.3.3 addresses feature selection.

5.3.1 Data Set Issues.

The data set proves extremely important to a machine learning scheme. In order to ensure that the trained model learns information regarding current malware behavior, the input data set malware must remain current. Theoretically, by including all malicious behaviors the behavior model includes all relevant information, and the model detects all germane malware.

Every application that exists on a given ECI contains executable files. Each additional behavior that contributes to a behavior model increases the ability of a model to correctly classify an executable file that exhibits similar behavior. Cyber defenders must identify applications present on a specific ECI and include representative executable files to enable appropriate model training. This methodology uses 32-bit Windows XP SP3 virtual guests, and includes operating system files from Windows 2000 to Windows 7. A current limitation of the Cuckoo Sandbox configuration pushes the ability to operate 64-bit guests outside the scope of this research, but the Cuckoo Sandbox developer intend to provide 64-bit capability in the future. In an operational environment, cyber defenders implementing this method must match the sandbox environment to the enterprise environment as closely as possible. Any differences between the tests and the operational workstations could mean that executable files behave differently, which could increase the chances of missing a malware sample in the wild.

The malware comes from a source that includes a wide variety of malware. Additional work to identify original sources of malware samples enables more accurate labeling and more specific classification training. Researchers could use method similar to this to attempt to categorically identify generic malicious behaviors that provide insight to the unique malicious behaviors that exist in the wild. This would enable building very accurate and efficient models.

5.3.2 Machine Learning Algorithms.

Different machine learning algorithms produce different types of models. Certain algorithmic heuristics discover the ideal information from the input data [33]. On the other hand, discovering a better performing learning algorithm for a given data set requires a substantial research effort. Decision trees are sufficient to classify whiteware and malware relative to a data set. Ensemble learning methods prove effective to improve adequate-performing learning algorithms [17]. Again, pilot studies for this research show that decision trees perform better than bagging or boosting decision stumps and similarly to bagging or boosting decision trees. Ensembles of decision stumps train more quickly than decision trees, but decision trees train faster than ensembles of decision trees because each ensemble trains 10 models internally. Future work could investigate ensembles of smaller decision trees than normal, but not as small as decision stumps, which just have one level.

Algorithms such as support vector machine (SVM) and perceptrons find applications in other fields, but decision trees attain high accuracy in malware classification without the need for the high computational cost of other robust algorithms.

5.3.3 Feature Selection.

The scope of these experiments covers 500 features according to information gain. While this number covers the useful distinctions between malware and non-malware with some probability, attaining higher probability requires additional features. Some samples do not exhibit any behavior that scores as high an information gain as the feature selection set. Then the feature vector for such a sample contains all zeros. This method discards such samples, so any behavior that the sample exhibits does not contribute to the detection model. Increasing the number of features decreases the number of thusly unrepresented samples. This decrease demonstrates a bias of the feature selection method for behaviors that a larger number of samples exhibit along classification lines. In this case, multiple behaviors that correctly correspond to classes identity the same group of samples, but a smaller group of samples does not contribute as highly.

Future study should investigate alternate feature selection schemes and parameters. Using principal component analysis could be a different way to rank the dimensions from the generated feature set. A differently-sized feature selection set could affect classifier performance. For example, including more than 500 features provides more information to the learning model. However, more input features requires more computational resources to build the model. One approach could be to select a varied number of features based on the number of *q*-grams that exist in the behavior reports.

However, this study shows that machine learning algorithms can learn an accurate model when the number of features is small compared to the number of possible features.

Appendix A: Top 10 Performance Results

Table A.1: Top 10 Data Sets by Classifier Performance according to the Area under ROC (AUC) at the 99.9% confidence interval (C.I.)

MIST	q	N	AUC	std. dev.	std. err.	99.9% C.I.
2	13	200	0.998137	0.001189	0.000084	± 0.000281
2	14	200	0.997943	0.001055	0.000075	± 0.000249
2	12	200	0.997883	0.001054	0.000075	± 0.000249
2	11	200	0.997817	0.001072	0.000076	± 0.000253
2	6	200	0.996767	0.001179	0.000083	± 0.000278
2	10	200	0.995889	0.003728	0.000264	± 0.000881
2	1	200	0.995805	0.003423	0.000242	± 0.000809
2	5	200	0.992861	0.003221	0.000228	± 0.000761
2	4	200	0.992662	0.002849	0.000201	± 0.000673
2	8	200	0.992503	0.006521	0.000461	± 0.001540

Table A.2: Top 10 Data Sets by Classifier Performance according to the Percent Correct at the 99.9% confidence interval (C.I.)

MIST	*q*	N	% correct	std. dev.	std. err.	99.9% C.I.
2	13	200	99.3819	0.131649	0.009309	± 0.0311
2	14	200	99.3696	0.120930	0.008551	± 0.0286
2	10	200	99.3255	0.120874	0.008547	± 0.0285
2	4	200	99.2708	0.110091	0.007785	± 0.0260
2	8	200	99.2180	0.142474	0.010074	± 0.0336
2	2	200	99.1946	0.114426	0.008091	± 0.0270
2	9	200	99.1835	0.127091	0.008987	± 0.0300
2	12	200	99.1808	0.151388	0.010705	± 0.0358
2	11	200	99.1552	0.167846	0.011868	± 0.0396
2	7	200	99.1536	0.144591	0.010224	± 0.0341

Table A.3: Significance of MIST Level by Factor Levels, sorted ascending by AUC of MIST level 1 from left to right then top to bottom

q	norm	MIST level 1	2		q	norm	MIST level 1	2	
1	full	97.7033	99.0775	○	1	norm	97.7273	99.0997	○
6	full	97.8300	99.0102	○	5	full	97.8723	99.0265	○
2	full	97.8810	99.1275	○	3	full	97.8850	98.8098	○
2	norm	97.9027	99.2616	○	5	norm	97.9083	99.1942	○
6	norm	97.9093	99.1538	○	7	full	97.9387	99.0953	○
3	norm	97.9478	98.9784	○	14	full	97.9734	99.3966	○
4	full	97.9979	99.2236	○	8	full	98.0128	99.1727	○
7	norm	98.0345	99.2119	○	9	full	98.0430	99.1759	○
4	norm	98.0480	99.3179	○	13	full	98.0738	99.3832	○
14	norm	98.0757	99.3427	○	12	full	98.0785	99.1723	○
11	full	98.1033	99.1149	○	10	full	98.1209	99.3015	○
13	norm	98.1301	99.3807	○	12	norm	98.1346	99.1893	○
8	norm	98.1516	99.2634	○	11	norm	98.1671	99.1956	○
9	norm	98.1979	99.1910	○	10	norm	98.2148	99.3495	○
15	full	98.9178	99.0332		16	full	98.9530	98.9768	
15	norm	99.0416	99.0569		16	norm	99.1217	98.9485	

○ statistically significant improvement, 0.001 significance

Table A.4: Significance of Normalization by Factor Levels, sorted ascending by AUC of Non-norm from left to right then top to bottom

MIST level	q	Non-norm	Normalized		MIST level	q	Non-norm	Normalized	
1	1	97.7033	97.7273		1	6	97.8300	97.9093	
1	5	97.8723	97.9083		1	2	97.8810	97.9027	
1	3	97.8850	97.9478		1	7	97.9387	98.0345	
1	14	97.9734	98.0757	∘	1	4	97.9979	98.0480	
1	8	98.0128	98.1516	∘	1	9	98.0430	98.1979	∘
1	13	98.0738	98.1301		1	12	98.0785	98.1346	
1	11	98.1033	98.1671		1	10	98.1209	98.2148	
2	3	98.8098	98.9784	∘	1	15	98.9178	99.0416	
1	16	98.9530	99.1217	∘	2	16	98.9768	98.9485	
2	6	99.0102	99.1538	∘	2	5	99.0265	99.1942	∘
2	15	99.0332	99.0569		2	1	99.0775	99.0997	
2	7	99.0953	99.2119	∘	2	11	99.1149	99.1956	
2	2	99.1275	99.2616	∘	2	12	99.1723	99.1893	
2	8	99.1727	99.2634		2	9	99.1759	99.1910	
2	4	99.2236	99.3179	∘	2	10	99.3015	99.3495	
2	13	99.3832	99.3807		2	14	99.3966	99.3427	

∘ statistically significant improvement, 0.001 significance

Appendix B: Experiment Result Details

This appendix includes the detailed results from the 64 experiments. The following list contains the definitions of the column headings in each of the four tables:

- norm – normalization
- *q* – gram length
- *l* – malware instruction set (MIST) level
- #tng – number of training samples
- #test – number of testing samples
- %acc – percent correct
- AUC – area under the receiver operating characteristic (ROC) curve
- FPR – false positive rate
- FNR – false negative rate
- TP – number of true positives
- FP – number of false positives
- TN – number of true negatives
- FN – number of false negatives
- tng (s) – user CPU time training (seconds)
- test (s) – user CPU time testing (seconds)
- size (b) – serialized model size (bytes)

Table B.1: Experiment Result Details for MIST level 1

	norm	q	l	#tng	#test	%acc	AUC	FPR	FNR	TP	FP	TN	FN	tng (s)	test (s)	size (b)
1	non-norm	1	1	77,886	8,654	97.703	0.9899	0.0750	0.0050	6,400	167	2,055	32	142	0.028	69,252
2	norm	1	1	77,886	8,654	97.727	0.9828	0.0763	0.0042	6,405	170	2,053	27	277	0.030	71,357
3	non-norm	2	1	77,461	8,607	97.881	0.9895	0.0683	0.0048	6,357	152	2,067	31	1,612	0.071	172,684
4	norm	2	1	77,461	8,607	97.903	0.9888	0.0672	0.0049	6,356	149	2,070	31	3,489	0.069	173,799
5	non-norm	3	1	77,254	8,584	97.885	0.9888	0.0715	0.0037	6,348	158	2,055	23	1,471	0.070	152,066
6	norm	3	1	77,254	8,584	97.948	0.9890	0.0664	0.0046	6,342	147	2,066	29	3,411	0.073	163,622
7	non-norm	4	1	76,233	8,470	97.998	0.9890	0.0673	0.0035	6,258	148	2,043	22	1,571	0.062	146,494
8	norm	4	1	76,233	8,470	98.048	0.9897	0.0650	0.0037	6,257	142	2,048	23	2,317	0.066	157,205
9	non-norm	5	1	67,028	7,448	97.872	0.9828	0.1170	0.0028	6,227	141	1,062	18	1,620	0.058	143,021
10	norm	5	1	67,028	7,448	97.908	0.9814	0.1134	0.0031	6,225	136	1,067	19	1,883	0.060	149,657
11	non-norm	6	1	66,470	7,386	97.830	0.9780	0.1263	0.0020	6,204	148	1,021	13	1,690	0.056	133,688
12	norm	6	1	66,470	7,386	97.909	0.9820	0.1155	0.0031	6,198	135	1,034	19	1,814	0.059	149,886
13	non-norm	7	1	65,854	7,317	97.939	0.9784	0.1185	0.0022	6,146	137	1,020	14	1,783	0.058	133,910
14	norm	7	1	65,854	7,317	98.034	0.9767	0.1113	0.0024	6,145	129	1,028	15	1,607	0.059	146,444
15	non-norm	8	1	64,881	7,209	98.013	0.9790	0.1156	0.0018	6,052	132	1,013	11	1,653	0.054	126,057
16	norm	8	1	64,881	7,209	98.152	0.9836	0.1054	0.0021	6,051	121	1,025	12	1,478	0.057	143,792

Table B.2: Experiment Result Details for MIST level 1, continued

	norm	q	l	#tng	#test	%acc	AUC	FPR	FNR	TP	FP	TN	FN	tng (s)	test (s)	size (b)
17	non-norm	9	1	62,696	6,966	98.043	0.9790	0.1132	0.0013	5,822	129	1,007	8	1,789	0.052	123,320
18	norm	9	1	62,696	6,966	98.198	0.9846	0.1018	0.0017	5,820	116	1,020	10	1,410	0.054	143,756
19	non-norm	10	1	60,069	6,674	98.121	0.9826	0.1038	0.0015	5,539	117	1,010	9	1,558	0.053	123,741
20	norm	10	1	60,069	6,674	98.215	0.9852	0.0965	0.0019	5,537	109	1,018	10	1,092	0.054	138,795
21	non-norm	11	1	59,737	6,637	98.103	0.9793	0.1048	0.0015	5,505	118	1,006	8	1,381	0.053	118,894
22	norm	11	1	59,737	6,637	98.167	0.9826	0.0985	0.0020	5,502	111	1,014	11	1,196	0.055	134,143
23	non-norm	12	1	59,387	6,599	98.079	0.9825	0.1049	0.0016	5,463	118	1,009	9	1,559	0.055	123,642
24	norm	12	1	59,387	6,599	98.135	0.9788	0.0993	0.0020	5,460	112	1,015	11	1,487	0.053	134,775
25	non-norm	13	1	55,840	6,204	98.074	0.9832	0.1003	0.0014	5,077	112	1,008	7	1,482	0.049	125,749
26	norm	13	1	55,840	6,204	98.130	0.9806	0.0947	0.0020	5,074	106	1,014	10	1,605	0.053	135,243
27	non-norm	14	1	55,400	6,156	97.973	0.9808	0.1043	0.0015	5,024	117	1,007	7	1,570	0.051	123,702
28	norm	14	1	55,400	6,156	98.076	0.9839	0.0953	0.0022	5,020	107	1,017	11	1,457	0.053	138,287
29	non-norm	15	1	34,005	3,778	98.918	0.9736	0.2714	0.0017	3,645	35	93	6	1,055	0.031	121,527
30	norm	15	1	34,005	3,778	99.042	0.9736	0.2005	0.0029	3,640	26	102	11	1,125	0.035	135,765
31	non-norm	16	1	30,568	3,396	98.953	0.9843	0.2444	0.0019	3,270	29	91	6	819	0.028	119,451
32	norm	16	1	30,568	3,396	99.122	0.9693	0.1778	0.0026	3,268	21	99	9	907	0.030	134,573

Table B.3: Experiment Result Details for MIST level 2

	norm	q	l	#tng	#test	%acc	AUC	FPR	FNR	TP	FP	TN	FN	tng (s)	test (s)	size (b)
33	non-norm	1	2	77,296	8,588	99.077	0.9990	0.0283	0.0026	6,357	63	2,152	17	740	0.060	131,250
34	norm	1	2	77,296	8,588	99.100	0.9926	0.0282	0.0023	6,359	63	2,152	15	2,177	0.061	127,770
35	non-norm	2	2	74,139	8,238	99.128	0.9892	0.0304	0.0008	6,031	67	2,135	5	1,539	0.052	118,841
36	norm	2	2	74,139	8,238	99.262	0.9955	0.0223	0.0019	6,024	49	2,153	12	1,788	0.053	136,126
37	non-norm	3	2	62,521	6,947	98.810	0.9885	0.0352	0.0012	4,752	77	2,112	6	885	0.053	117,432
38	norm	3	2	62,521	6,947	98.978	0.9919	0.0251	0.0034	4,742	55	2,134	16	1,497	0.046	141,558
39	non-norm	4	2	54,969	6,108	99.224	0.9903	0.0202	0.0010	3,950	44	2,111	4	762	0.044	112,355
40	norm	4	2	54,969	6,108	99.318	0.9950	0.0148	0.0025	3,944	32	2,122	10	1,067	0.045	128,744
41	non-norm	5	2	45,515	5,057	99.027	0.9907	0.0389	0.0010	3,888	45	1,120	4	608	0.036	113,886
42	norm	5	2	45,515	5,057	99.194	0.9950	0.0264	0.0026	3,882	31	1,135	10	882	0.034	130,441
43	non-norm	6	2	43,935	4,882	99.010	0.9959	0.0382	0.0014	3,748	43	1,085	5	379	0.036	115,444
44	norm	6	2	43,935	4,882	99.154	0.9976	0.0294	0.0022	3,745	33	1,095	8	404	0.038	127,407
45	non-norm	7	2	39,583	4,398	99.095	0.9783	0.0519	0.0011	3,704	36	654	4	552	0.035	116,984
46	norm	7	2	39,583	4,398	99.212	0.9876	0.0414	0.0016	3,702	29	662	6	622	0.030	127,337
47	non-norm	8	2	38,879	4,320	99.173	0.9939	0.0460	0.0011	3,627	32	657	4	490	0.032	118,062
48	norm	8	2	38,879	4,320	99.263	0.9911	0.0377	0.0016	3,626	26	663	6	587	0.032	128,232

Table B.4: Experiment Result Details for MIST level 2, continued

	norm	q	l	#tng	#test	%acc	AUC	FPR	FNR	TP	FP	TN	FN	tng (s)	test (s)	size (b)
49	non-norm	9	2	38,770	4,308	99.176	0.9896	0.0497	0.0005	3,630	34	642	2	522	0.032	118,446
50	norm	9	2	38,770	4,308	99.191	0.9908	0.0416	0.0019	3,625	28	648	7	546	0.032	130,030
51	non-norm	10	2	37,300	4,144	99.301	0.9981	0.0403	0.0008	3,491	26	624	3	499	0.030	116,430
52	norm	10	2	37,300	4,144	99.350	0.9937	0.0342	0.0014	3,489	22	628	5	538	0.035	127,387
53	non-norm	11	2	33,821	3,758	99.115	0.9979	0.0437	0.0015	3,100	29	624	5	356	0.027	116,966
54	norm	11	2	33,821	3,758	99.196	0.9977	0.0338	0.0026	3,097	22	631	8	410	0.028	127,250
55	non-norm	12	2	33,706	3,745	99.172	0.9981	0.0430	0.0010	3,092	28	622	3	441	0.026	116,642
56	norm	12	2	33,706	3,745	99.189	0.9977	0.0336	0.0028	3,086	22	628	9	407	0.027	126,014
57	non-norm	13	2	32,507	3,612	99.383	0.9987	0.0274	0.0016	2,966	18	623	5	413	0.027	116,466
58	norm	13	2	32,507	3,612	99.381	0.9976	0.0203	0.0031	2,962	13	628	9	378	0.026	125,097
59	non-norm	14	2	32,517	3,613	99.397	0.9984	0.0250	0.0019	2,964	16	627	6	393	0.026	115,654
60	norm	14	2	32,517	3,613	99.343	0.9974	0.0191	0.0039	2,959	12	631	11	461	0.025	120,787
61	non-norm	15	2	18,627	2,070	99.033	0.9746	0.5073	0.0012	2,033	18	17	2	88	0.014	115,640
62	norm	15	2	18,627	2,070	99.057	0.9632	0.4230	0.0024	2,030	15	20	5	81	0.015	120,585
63	non-norm	16	2	16,510	1,834	98.977	0.9746	0.5239	0.0008	1,800	17	16	1	53	0.012	114,027
64	norm	16	2	16,510	1,834	98.948	0.9736	0.4742	0.0020	1,798	16	17	4	45	0.012	119,174

Bibliography

[1] Arbor malware library (AML), 2006. URL http://www.arbornetworks.com/.

[2] Bailey, M., J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. “Automated Classification and Analysis of Internet Malware”. *Recent Advances in Intrusion Detection*, 178–197. Springer, 2007.

[3] Bayer, U., P.M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. “Scalable, behavior-based malware clustering”. *Network and Distributed System Security Symposium (NDSS)*. Citeseer, 2009.

[4] Bayer, U., A. Moser, C. Kruegel, and E. Kirda. “Dynamic analysis of malicious code”. *Journal in Computer Virology*, 2(1):67–77, 2006.

[5] Bayer, Ulrich and the International Secure Systems Lab. “Anubis: Analyzing Unknown Binaries”, May 30 2012. URL http://anubis.iseclab.org/.

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

[7] Dube, T.E. *A Novel Malware Target Recognition Architecture for Enhanced Cyberspace Situation Awareness*. Ph.D. thesis, Air Force Institute of Technology (AFIT), 2011.

[8] Duda, RO, PE Hart, and DG Stork. “Pattern classification”. 2001.

[9] Egele, Manuel, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. “A Survey on Automated Dynamic Malware Analysis Techniques and Tools”. *ACM Computing Surveys*, 2012.

[10] Eskandari, M. and S. Hashemi. “A graph mining approach for detecting unknown malwares”. *Journal of Visual Languages & Computing*, 2012.

[11] Firdausi, I., C. Lim, A. Erwin, and A.S. Nugroho. “Analysis of machine learning techniques used in behavior-based malware detection”. *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, 201–203. IEEE, 2010.

[12] Guarnieri, Claudio and Cuckoo Sandbox Developers. “Automated Malware Analysis - Cuckoo Sandbox”, Nov 15 2012. URL http://www.cuckoosandbox.org/.

[13] Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. “The WEKA Data Mining Software: An Update”. *SIGKDD Explorations*, Volume 11(Issue 1), 2009.

[14] Hu, X. *Large-Scale Malware Analysis, Detection, and Signature Generation*. Ph.D. thesis, The University of Michigan, 2011.

[15] Jackson, G.M. *Predicting Malicious Behavior: Tools and Techniques for Ensuring Global Security*. Wiley, 2012.

[16] Kolter, J.Z. and M.A. Maloof. "Learning to detect and classify malicious executables in the wild". *The Journal of Machine Learning Research*, 7:2721–2744, 2006.

[17] Kumar, G., K. Kumar, and M. Sachdeva. "Artificial Intelligence Based Ensembles for Intrusion Detection-A Study". *Journal of Computing Technologies*, 2012.

[18] Lindorfer, M., C. Kolbitsch, and P. Milani Comparetti. "Detecting environment-sensitive malware". *Recent Advances in Intrusion Detection*, 338–357. Springer, 2011.

[19] Lindorfer, Martina. *Detecting environment-sensitive malware*. Master of science in software engineering and internet computing, Vienna University of Technology, A-1040 Vienna, Karlsplatz 13, April 11 2011.

[20] Moser, A., C. Kruegel, and E. Kirda. "Exploring multiple execution paths for malware analysis". *Security and Privacy, 2007. SP'07. IEEE Symposium on*, 231–245. IEEE, 2007.

[21] Moser, A., C. Kruegel, and E. Kirda. "Limits of static analysis for malware detection". *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 421–430. IEEE, 2007.

[22] Oracle. "Oracle VM VirtualBox", Nov 15 2012. URL https://www.virtualbox.org/.

[23] Quinlan, J.R. "Induction of decision trees". *Machine learning*, 1(1):81–106, 1986.

[24] Quinlan, Ross. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[25] Quist, Daniel. "Open Malware", Oct 30 2012. URL http://offensivecomputing.net/.

[26] Rieck, K., P. Trinius, C. Willems, and T. Holz. "Automatic analysis of malware behavior using machine learning". *Journal of Computer Security*, 19(4):639–668, 2011.

[27] Rossow, C., C.J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Steen. "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook". *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 65–79. IEEE Computer Society, 2012.

[28] Scan4You.net. "Scan4You.net - Online Anonymous Virus and Malware Scan", Jan 6 2013. URL http://scan4you.net/.

[29] Trinius, P., C. Willems, T. Holz, and K. Rieck. “A malware instruction set for behavior-based analysis”. 2011.

[30] US-CERT. “US-CERT - United States Computer Emergency Readiness Team”, Jul 15 2012. URL http://www.us-cert.gov/.

[31] VirusTotal, Google. “VirusTotal - Free Online Virus, Malware and URL Scanner”, Jan 4 2013. URL http://www.virustotal.com/.

[32] Witten, I.H. and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[33] Wolpert, D.H. and W.G. Macready. “No free lunch theorems for optimization”. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[34] Yin, H., D. Song, M. Egele, C. Kruegel, and E. Kirda. “Panorama: capturing system-wide information flow for malware detection and analysis”. *Proceedings of the 14th ACM conference on Computer and communications security*, 116–127. ACM, 2007.

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

1. REPORT DATE *(DD–MM–YYYY)* 21–03–2013	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Oct 2011–Mar 2013

4. TITLE AND SUBTITLE Learning Enterprise Malware Triage from Automatic Dynamic Analysis	**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Bristow, Jonathan S., Captain, USAF	**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-ENG-13-M-10
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of Homeland Security ICS-CERT POC: Neil Hershfield, DHS ICS-CERT Technical Lead ATTN: NPPD/CS&C/NCSD/US-CERT Mailstop: 0635, 245 Murray Lane, SW, Bldg 410, Washington, DC 20528 Email: ics-cert@dhs.gov phone: 1-877-776-7585 Grant Number: HSHQDC-11-X-00089	**10. SPONSOR/MONITOR'S ACRONYM(S)** DHS ICS-CERT **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

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

14. ABSTRACT
Adversaries employ malware against victims of cyber espionage with the intent of gaining unauthorized access to information. To that end, malware authors intentionally attempt to evade defensive countermeasures based on static methods. This thesis analyzes a dynamic analysis methodology for malware triage that applies at the enterprise scale. This study captures behavior reports from 64,987 samples of malware randomly selected from a large collection and 25,591 clean executable files from operating system install media. Function call information in sequences of behavior generate feature vectors from behavior reports from the files. The results of 64 experiment combinations indicate that using more informed behavior features yields better performing models with this data set. The decision tree classifier attained a max performance of 0.999 area under the ROC curve and 99.4% accuracy using argument information with function sequence lengths from 11–14.

This methodology contributes to strategic cyber situation awareness by fusion with fast malware detection methods, such as static analysis, to change the game of malware triage in favor of cyber defense. This method of triage reduces the number of false alarms from automatic analysis that allows a 97% workload reduction over using a static method alone.

15. SUBJECT TERMS
malware classification behavior 'machine learning' dynamic analysis' 'cyber situation awareness' 'feature generation' 'feature selection' 'malware instruction set' 'n-gram' 'q-gram'

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Thomas E. Dube, Maj, USAF (ENG)
					19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	75	(937) 255-3636 ext. 4613