Feature Fusion Across Domains for Improved Malware Classification

A Thesis

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

Charles LeDoux

Fall 2011

Feature Fusion Across Domains for Improved Malware Classification

Charles LeDoux

APPROVED:

Arun Lakhotia, Chair Professor of Computer Science The Center for Advanced Computer Studies	Andrew Walenstein Assistant Professor of Computer Science
Anthony Maida Associate Professor of Computer Science The Center for Advanced Computer Studies	David Breaux Dean of the Graduate School

ACKNOWLEDGMENTS

I would like to thank the many people whose suggestions and guidance made this thesis possible. First, I would like the thank my advisor, Dr. Arun Lakhotia, who has guided me through this arduous journey of thesis writing. His feedback on both the concepts of this thesis and my writing style were invaluable. Next I would like to thank Dr. Andrew Walenstein for continually helping me refine the formalisms and the organization of this thesis. I would also like to thank Craig Miles for assisting me in understanding the complex world of statistics and Daniel Hefner who aided in the preparation and collection of the data. I would also like to acknowledge the funding agencies which partially supported this work: the Air Force Research Lab, DARPA (FA8750-10-C-0171), and the Air Force Office of Scientific Research (FA9550-09-1-0715). Finally, I'd like to thank my wife, Hayley LeDoux, who has supported me in my academic endeavors and whose support I would be unable to continue without.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation

In order to make the life of a malware analyst difficult, malware authors often attempt to prevent analysis by utilizing different types of obfuscations. The term "obfuscation" can be used to refer to any method or technique used to prevent correct or complete data collection with respect to the obfuscated program. For example, in malware analysis, a debugger is often used to determine the runtime activities of the analyzed program. Thus, one obfuscation often used by malware authors is to check and see if a debugger is running and if it is, then the malicious activity is kept from performing. Since the data collected by the debugger is incomplete, any analysis based on this data will also be incomplete and possibly inaccurate.

A classic work in past obfuscation research showed that creating a universal obfuscation which works against all data collection techniques is not possible [1]. Thus any obfuscation is, by necessity, targeted towards a limited number of data collection techniques. For example, implementing a debugger check may prevent debugger based data collection, but it will not directly prevent accurate disassembly. Even techniques which measure similar properties can have unrelated weaknesses. For example, a linear sweep disassembler can be defeated using junk byte insertions, while a recursive traversal disassembler is instead defeated by opaque predicates [2]. This raises the question: can information collected using different techniques be combined in a way that turns the weaknesses of these techniques into a strength?

There is an area of research known as Information Fusion which focuses on the

problem of combining information in meaningful and useful ways. A simple, biological example of Information Fusion can be found in the phrase "Stop, look, and listen." What this phrase instructs us to do is to stop before crossing the street; then look *and* listen for traffic. This combination of visual and audio information improves the reliability of information regarding the safety of crossing the street; thus improving the reliability of the decision regarding the safety of crossing the street.

One type of Information Fusion, feature fusion across domains, is well suited for combining information collected about malware. Feature fusion across domains combines information collected from "sensors" which operate "across domains." While the term "sensor" is generally used to mean a sensor of physical data, the term is adapted here to mean any collector of data. The term "across domains" means that the sensors collect similar information in different ways and thus do not share the same weaknesses. For example, a motion sensor and a laser trip wire both collect information regarding the presence of an intruder, but they collect this information using different methods. By combining information on obfuscated malware obtained from sensors operating on different domains, it is expected that patterns of "missing" features, hereafter called obfuscation patterns, can be generated to aid in malware analysis. For example, if sensor A measures many properties about a malicious executable, but sensor B measures very little, it is a reasonable assumption that the malicious executable was obfuscated against sensor B.

The obfuscation patterns generated by feature fusion across domains are of particular interest to malware classification. Malware authors do not often write completely new malicious programs. Instead, they often modify one of their already existing malicious files, either to add a new functionality or to avoid detection. Malware analysts use classification to group malware into "families" of these variants. If the malware analyst can determine the family of a previously unseen malware, then he does

not have to analyze this file *ex novo*, but can rather reuse knowledge gained from analysis of other malware in the same family. Since malware in the same family contain similar behavior, it is expected that they will also contain similar obfuscations and thus similar obfuscation patterns. Consequently, if these obfuscation patterns are presented to the classifier, it is expected that the accuracy of classification will improve.

In order to present the obfuscation patterns to the classifier, there must exist a method of combining the information which encodes these patterns. Currently, feature fusion performed in malware analysis uses union as its fusion mechanism [3, 4, 5]. By simply combining features, the obfuscation patterns created by fusion across domains are lost. A new fusion mechanism is needed which uses rather than discards this additional information.

1.2. Research Objectives

The research objectives of this thesis are as follows:

- Propose a new feature fusion method generally useful for the sensors typically used in malware analysis. Specifically, the fusion should improve the accuracies of malware classifiers built using this sensor data.
- Quantify the expected benefits of the proposed method of fusion. There are a number of different learning algorithms which can be used to build a classifier, each operating in fundamentally different ways. We say that fusion is useful if it is able to increase the accuracy obtained by a meaningful fraction of classifiers in a statistically significant manner.

1.3. Research Contributions

There are two contributions presented in this thesis:

1. A new fusion function is proposed that takes advantage of the obfuscation patterns created by fusion across domains. This fusion function is known as a "disjoint union" in set theory. A disjoint union unions the elements of multiple sets, but unlike regular union, it records the originating set of every item. Similarly, feature fusion by disjoint union combines features while remembering which sensor detected which feature.

2. Empirical data that validates our proposed approach and quantifies its benefits is presented. An experiment was performed that used four different learning algorithms and over 18,000 malicious executables. The usefulness of feature fusion across domains is evaluated by testing classification accuracies using fusion of behavioral features from multiple behavioral extractors over classifications done with no fusion. The hypothesis that fusion by disjoint union provides additional information to the classifier is tested by comparing it to fusion by union. The experimental data suggests that disjoint union is generally useful. Further, disjoint union consistently outperforms fusion by union, implying that disjoint union does indeed provide additional information to the classifier.

1.4. Organization of Thesis

This thesis is organized as follows. In Chapter 2 background information is given. Chapter 3 motivate and proposes feature fusion across domains using disjoint union. Chapter 4 reports on the experimental evaluation of the proposed fusion method. Finally, concluding remarks are given in Chapter 5.

CHAPTER 2

BACKGROUND

This chapter provides background information on Information Fusion and malware classification and enumerates current applications of Information Fusion to malware classification

2.1. Information Fusion

Hall and Llinas [6] proposed a taxonomy of Information Fusion types that distinguishes three different types: data fusion, feature fusion, and decision fusion. Each of these types corresponds to the level in the decision process in which the fusion occurs. Data fusion is a combination of the raw outputs of the sensors before any preprocessing or feature extraction. This can only be done if the form (syntax) of the data is similar. Feature fusion occurs on features extracted from the raw data. It is also possible to delay fusion until the very end of the process, where the decision or action occurs. This is called decision fusion.

According to the taxonomy of Bob and Frank [7], each of the three types of information fusion can be done one of four ways: across sensors, across attributes, across domains, or across time. Fusion is across sensors when the information for fusion is obtained from multiple sensors measuring the same type of property. This is usually done to provide a more complete picture (point two cameras in slightly different directions so the whole room is recorded) or for fault tolerance (point the cameras in the same direction so that even if one fails, the room is still recorded). Fusion across attributes is the combining of information obtained from sensors measuring different properties (a visual camera and an audio mic). Fusion is across domains when the

information being fused is obtained from sensors measuring the same type of properties, but from different domains or at different levels. For example, both visual and infra-red cameras measure light, but they measure different spectrums. Another example is motion sensors and laser trip wires. They both measure information regarding the presence of an intruder, but in completely different ways. Finally, fusion across time is the fusion of information collected from a single sensor at different points in time.

At the heart of information fusion is the fusion function, that is the method used to combine the information in a useful and meaningful way. The choice of fusion function is entirely dependent on the specific application of information fusion. Nakamura et al. [8], outlines several of the more commonly used fusion functions for classification applications. These functions are: Bayesian Inference, Dempster-Shafer Inference, fuzzy logic, and neural networks. Bayesian Inference utilizes Bayes' rule [9] to assign probabilities to values. Dempster-Shafer Inference is a generalization of Bayesian Inference from probabilities to beliefs or mass functions. Fuzzy logic is a way to handle approximate reasoning and imprecise information. Neural networks are supervised learning algorithms which are able to generalize from a set of examples.

The fusion functions listed above focus on combining multiple values in order to produce a single value of the same type. This is not the focus of the fusion performed in this thesis. Rather, the focus is taking multiple values and combining them into a single value of a new type which contains all the information present in the combined values, plus additional information due to the fusion. Thus these classic fusion functions cannot be used and a new one must be defined.

2.2. Malware Classification

There are three general phases in malware classification (or any classification) which perfectly align with the three types of information fusion. These phases are: data

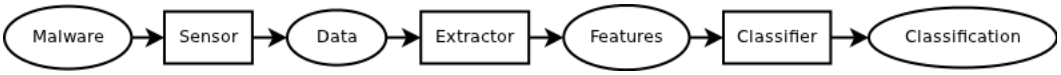

Figure 2.1. Phases of classification

collection, feature extraction, and classification. Figure 2.1 illustrates these phases. First, a malware file is submitted to a sensor. This sensor outputs analysis data and from this raw sensor data features are extracted. These features are given to the classifier which assigns a class label to the malicious file.

Malware sensors can be divided into two broad categories: static and dynamic. Static sensors are sensors which operate on the "static" text of the program. The raw data is obtained without running the program. A common static sensor is a disassembler. A disassembler takes as input a binary file and produces as output an assembly model of the program. Dynamic sensors, on the other hand, are sensors which measure "dynamic" properties of malware by executing the malicious file. A debugger is a commonly used dynamic sensor. A debugger watches a program as it executes and is capable of recording different information about the activities of the watched program. Normally, a trace of the instructions executed or the system calls made is recorded.

From the sensor data, features are extracted to represent the malware file. Features, like sensors, can be divided into static and dynamic features. A classic static feature is binary n-grams. N-grams are defined as a subsequence of length n from a larger sequence. The string 'abacd', for example, has the 2-grams 'ab', 'ba', 'ac', and 'cd'. Binary n-grams, then, are n-length sequences of bytes present in the malicious file. A commonly used dynamic feature, also called behavioral feature, is system calls. The sequence of system calls made by an executable is normally a fair indicator of the type of activities performed by the executable.

Once the features of a malware file are extracted, classification can begin. In machine learning based classification, a classifier must be constructed from a

Work	Fusion Method	Fusion Purpose
Lu et al. [3]	Union of feature sets	Use both static and dynamic features
Lu et al. [3]	Cascade SVM and AR Classifiers	Improve the accuracy of SVM by checking classification with AR
Masud et al. [10]	Union feature sets	Combine features representing knowledge from different abstraction layers
Islam et al. [5]	Union feature sets	Combine two already useful feature sets into a more useful feature set
Biggio et al. [11]	Randomize weights assigned to a multiple classifier ensemble	Hide information from an adversary
Menahem et al. [12]	Several known methods of combining classifiers	Determine optimal method of creating classifier ensemble
(this thesis)	Disjoint union of feature sets	Encode obfuscation patterns for the classifier

Table 2.1. Applications of Information Fusion in Malware Classification

representative data set before class label assignment (classification) can occur. To construct a classifier, a number of malware files who's class labels are already known (called the labeled data set) are selected. These files are submitted to the sensor and their features extracted. Each extracted feature set is labeled with the class the malware file belongs to. These labeled feature sets are given as input to a learning algorithm which gives as output a classifier. This classifier then performs classification on any unlabeled feature set presented to it.

2.3. Related Work

There is currently no literature in malware classification using feature fusion across domains. There are examples, however, of other types of fusion being applied for various reasons in malware classification. Table 2.1 summarizes these examples and gives the reasons fusion was applied.

Lu et al. [3], uses two different types of fusion to accomplish different things. First, he uses feature fusion across attributes to combine static and dynamic features, instead of using only one type. The static features chosen were the names of system calls present in the binary of the executable and the dynamic features were expert defined

behaviors commonly seen in malicious programs. Several examples of the behaviors searched for are packing the executable, DLL injection, and hiding files. There were twelve such features defined. These static and dynamic feature sets were fused by taking the union of the feature sets.

In addition to combining static and dynamic features, Lu et al. [3] also used decision fusion to improve the accuracy of classification. He began with a Support Vector Machine (SVM) and cascaded it with an Association Rule (AR) classifier. To build the cascaded classifier, called SVM-AR, a SVM was built to classify executables as either malicious or benign and two sets of Association Rules were assembled - one to test if an executable was malicious and one to test if it was benign. To classify an executable, it was first given to the SVM and then tested against the Association Rule opposite to the label given by the SVM. So, for example, if the SVM labeled an executable as malicious, it would be tested using the Association Rule for benign. If this rule matched, then the executable was classified as benign, else it was classified as malicious.

Masud et al. [10] used feature fusion across attributes to create a hybrid feature set containing features from three different levels of abstraction. The three features used were binary n-grams, derived assembly features (DAF), and system calls. Binary n-grams are all the n-length sequences of bytes present in a binary and thus are the features at the lowest level of abstraction. The feature on the next level of abstraction is the DAF. A DAF is the assembly instruction sequence from the executable which contains the bytes present in a given binary n-gram. At the highest level of abstraction are system calls. The names of the system calls present in the header of the executable were extracted and used.

Islam et al. [5] uses feature fusion to combine two useful feature sets in an attempt to make an even more useful feature set. He takes the union of function length frequency and printable string feature sets. Function length frequency is the number of

functions a program has of a given length. It was found by Islam that malware within the same family tend to have functions of around the same length, thus motivating the use of function length frequency as a feature. Printable strings are simply sequences of bytes which can be interpreted as valid ASCII strings.

Biggio et al. [11] addressed the adversarial nature of malware classification through decision fusion across domains. Adversarial classification is classification done against an intelligent adversary who is actively attempting to prevent correct classification. To take active steps against classifiers, something about the classifier must be known. Biggio attempted to hide critical knowledge from the adversary by introduction an element of randomness to the classification algorithm. To do this, he select multiple classifiers, performed classifications in parallel, and randomly assigned weights to the resulting classifications. These random weights were then used to determine the final classification.

Menahem et al. [12] performed a study of several different decision fusion techniques to determine the optimal fusion function. He compared the following fusion functions: best classifier, majority voting, performance weighting, distribution summation, Bayesian combination, Naïve Bayes, stacking, and a method he created: troika. In the best classifier fusion function, the classifier which performs the best is chosen. Majority voting assigns the class which the majority of classifiers agree on. Performance weighting assigns a weight to the output of each classifier based upon its accuracy against some test data set. Distribution summation works by summing up the conditional probability vector obtained from each classifier and assigning the class with the highest total value in the vector. Bayesian combination assigns a weight to each classifier based on the posterior probability of the classifier given the training set. Naïve Bayes uses applications of Bayes' rule. Stacking attempts to learn which classifiers are reliable and which are not by building a meta-classifier. This meta-classifier produces a

final classification based on the class assignments of the individual classifiers. Finally, troika [13] is a generalization of stacking which is capable of combining any type of classifiers.

The common thread among all the above listed examples of fusion in malware classification is a desire to increase the accuracy of the classification. Each work has approached this overall goal from a different angle and improved one facet of or solved one problem in malware classification. In [3, 11, 12] the focus was on improving the classifier itself; increasing the quality of the decisions made. In [3, 10, 5], however, the focus was on improving the feature sets. These works tried to increase the type and amount of information given to the classifier. Similarly, this thesis attempts to increase the information present in the feature sets, but by creating new information through fusion rather than simply combining information already present in differing feature sets.

CHAPTER 3

FEATURE FUSION ACROSS DOMAINS USING DISJOINT UNION

I propose combining information obtained from multiple sensors to generate obfuscation patterns and thus provide additional information to the classifier in order to increase the accuracy of classification. In order to combine this information, the type and method of fusion must be decided upon and a fusion function created. I propose feature fusion across domains as the fusion method and type and disjoint union as the fusion function.

3.1. Feature Fusion Across Domains

The fusion type used to create obfuscation patterns is feature fusion. Obfuscation patterns are the patterns of "missing" information between different sensor data. Thus, the necessary information for generating these patterns is stored within the sensor data. However, it is not always possible to directly combine sensor outputs. Often, their "syntaxes" are not compatible. Thus, it is necessary that features are extracted from the data before fusion can occur. Additionally, since the obfuscation pattern information is present in the differences between sensor data, it is not present at the decision level. Thus, feature fusion is the best choice for fusion type. Figure 3.1 illustrates the phases of classification when feature fusion is present. The only difference from when no fusion is used is the presence of multiple sensors and a fuser.

The fusion method used is fusion across domains. In order for obfuscation patterns to be generated by fusing feature sets, there are two requirements. The first is there must be some expected overlap between the feature sets. A "missing" feature is only detected when it is supposed to be in the feature set for both sensor A and sensor B, but it is only present in one of them. If the feature sets for sensors are always mutually

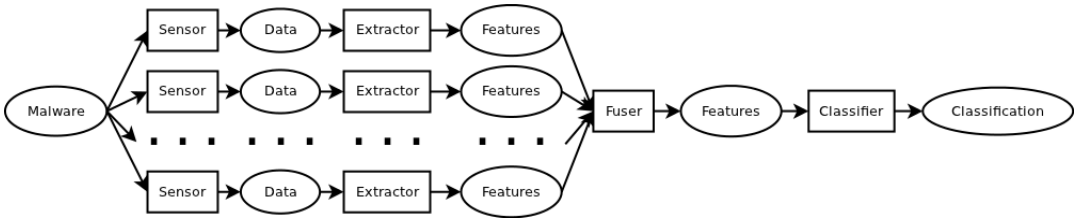

Figure 3.1. Phases of classification with feature fusion

exclusive, then this "missing" feature detection is not possible. The second requirement is that the sensors must not share the same weaknesses. This is also necessary for "missing" feature detection. A "missing" feature cannot be detected if the obfuscation defeated (prevented data collection) all sensors equally as the feature will not be present in any of the feature sets.

3.2. Fusion by Disjoint Union

I propose to use "disjoint union" as the fusion function to generate obfuscation patterns. In set theory, a disjoint union is a union which combines the sets, but records the originating set of every item in the union set. Formally, let $\{A_i : i \in I\}$ be a family of sets indexed by I. Then the disjoint union of the family of sets is defined as $\bigsqcup_{i \in I} A_i = \bigcup_{i \in I} \{(x, i) : \forall x \in A_i\}$. The resulting set contains the ordered pairs (x, i) where x is an element of set i. Figure 3.2 gives an example of disjoint union. The sets A_1, A_2, A_3 contain the elements $\{a, b, c\}$, $\{b, c\}$, and $\{c, d\}$ respectively. The disjoint union of these three sets is the set $\{(a, 1), (b, 1), (c, 1), (b, 2), (c, 2), (c, 3), (d, 3)\}$ which indicates that element a was in set 1 only, element b was in sets 1 and 2, element c was in all three sets, and element d was only in set 3.

Similarly, fusion by disjoint union combines feature sets extracted from multiple sensors in such a way that which sensor detected which feature is recorded. Adapting the set theory definition of disjoint union, let $\{S_i : i \in I\}$ be the sets of features

$$
\begin{aligned}
&A_1 = \{a, b, c\} \\
&A_2 = \{b, c\} \\
&A_3 = \{c, d\} \\
&A_1 \bigsqcup A_2 \bigsqcup A_3 = \{(a,1), (b,1), (c,1), (b,2), (c,2), (c,3), (d,3)\}
\end{aligned}
$$

Figure 3.2. Example of disjoint union

extracted from the sensors indexed by I for a malware file. Then, fusion by disjoint union is defined as $\bigsqcup_{i \in I} = \bigcup_{i \in I} \{(f, i) : \forall f \in S_i\}$. This gives us a set containing the ordered pairs (f, i) where f is a feature extracted from the output of sensor i. Figure 3.3 gives an example of feature fusion by disjoint union. There are three sensors: S_1, S_2, and S_3. From the output of sensor S_1, features f_1 and f_2 were extracted. From the output of sensor S_2, features f_2 and f_3 were extracted. Only feature f_1 was extracted from the output of sensor S_3. Feature fusion by disjoint union creates the feature set $\{(f_1, 1), (f_2, 1), (f_2, 2), (f_3, 2), (f_1, 3)\}$. This feature set indicates that feature f_1 was extracted from the outputs of sensors S_1 and S_3, feature f_2 was extracted from the output of sensors S_1 and S_2, and feature f_3 was only extracted from the output of sensor S_2. The disjoint union is, in effect, creating an inverted index for each feature listing the sensors from which the feature was obtained.

$$
\begin{aligned}
S_1 &= \{f_1, f_2\} \\
S_2 &= \{f_2, f_3\} \\
S_3 &= \{f_1\} \\
F &= S_1 \bigsqcup S_2 \bigsqcup S_3 \\
&= \{(f_1, 1), (f_2, 1), (f_2, 2), (f_3, 2), (f_1, 3)\}
\end{aligned}
$$

Figure 3.3. Example of feature fusion by disjoint union

CHAPTER 4

EVALUATION

In order to evaluate the usefulness of using feature fusion across domains by disjoint union for malware classification, an experiment is conducted where the fusion of features acts as the independant variable and the accuracy of classifications acts as the dependant variables. This experiment is repeated with a number of different classifiers to minimize the effects of any bias present in the learning algorithms. K-Fold Cross Validation is used to generate the accuracy measurement and the results are analyzed for statistical significance using paired t-test and confidence intervals.

There are two hypotheses being tested:

1. Feature fusion across domains using disjoint union increases the accuracy of classification.

2. Disjoint Union provides additional information to the classifier related to the patterns of "missing" features created by obfuscations.

The first hypothesis is tested by comparing the accuracies obtained when no feature fusion is used to the accuracies obtained when feature fusion is used. It is theorized that classifications performed using feature fusion will have statistically significant higher accuracies.

The second hypothesis is tested by comparing accuracies obtained using disjoint union as the fusion function to accuracies obtained using union as the fusion function. If disjoint union does not provide any additional information to the classifier, then its accuracies should be very similar to the accuracies of union. Thus, if disjoint union

attains significanly higher accuracies than union, it can be concluded that disjoint union does indeed provide additional information to the classifier.

4.1. Apparatus

The experimental evaluation requires generation of feature sets, fusion of these feature sets, and performing classifications using both the fused and non-fused feature sets. To accomplish this, sensors which operate on different domains are chosen, classifiers are selected, and a mechanism to implement the fusion is put into place.

4.1.1. Sensors

The sensors chosen were two web based behavioral analysis systems, Anubis (formerly TTAnalyze) [14] and Cwsandbox [15]. The way these systems work is the sample to be analyzed is submitted through a web interface; the system performs a single execution of the malware sample; information on what the sample does (its behavior) is collected during execution; and an XML report listing the detected behaviors is returned.

These two sensors were chosen because the contain the properties necessary for fusion to be across domains: there is expected overlap between the feature sets and the methods of data collection are different enough so that the sensors have dissimilar weaknesses. Both of these sensors are meant to detect operating system level activities. These are actions such as creating or editing files, loading a DLL, reading the value in a registry key, etc. These two sensors collect data differently by gathering information at different levels. The different levels they operate on are illustrated in Figure 4.1.

Typically, user level programs do not directly interact with the hardware. Rather, the operating system defines a set of functions known as the Application Binary Interface (ABI) which user level programs can utilize in order to perform operations on the hardware. The functions defined in the ABI are known as system calls and calling

one of these functions is known as making a system call. So to access a file, for example, a user program would make a system call by calling the function in the ABI defined for accessing files. The operating system would then check to make sure that the user program has the appropriate privileges, and if it does, creates a file handle for the requested file and returns it as the return value of the system call. The user program can then access the contents of the file.

Anubis runs in a modified Qemu and works by inserting itself between the operating system and the hardware [16]. Instead of the operating system directly communicating with and controlling the hardware, it instead communicates with the virtualized hardware provided by Anubis through Qemu. All instructions executed by or in the operating system, including instructions executed by programs in the operating system, are executed on the virtual hardware provided by Anubis. Anubis looks through this dynamic instruction trace to find the behaviors of the programs to be analyzed. In order to only analyze the desired program and not the entire operating system, the CR3 register, also called the page-directory base register (PDBR), is used. The CR3 register contains the physical address of the base of the page directory of the currently executing process. Since Windows assigns each process a unique page directory, the value in the CR3 register uniquely identifies the currently executing process. Thus, by determining the address of the base of the page directory of the analyzed program, virtually all the noise from the operating system can be removed from the instruction trace.

Cwsandbox inserts itself between the ABI and the operating system by utilizing a technique known as DLL hooking [17]. In Windows, the ABI is contained in a collection of DLL files. Cwsandbox knows which DLLs contain the Window's ABI and whenever these DLLs are loaded into memory, Cwsandbox modifies the system calls in them so that whenever a system call is made, the control flow is first "hooked" and redirected to Cwsandbox instead of continuing to the operating system. After Cwsandbox records

any information it needs, it returns the control flow back to where it left off.

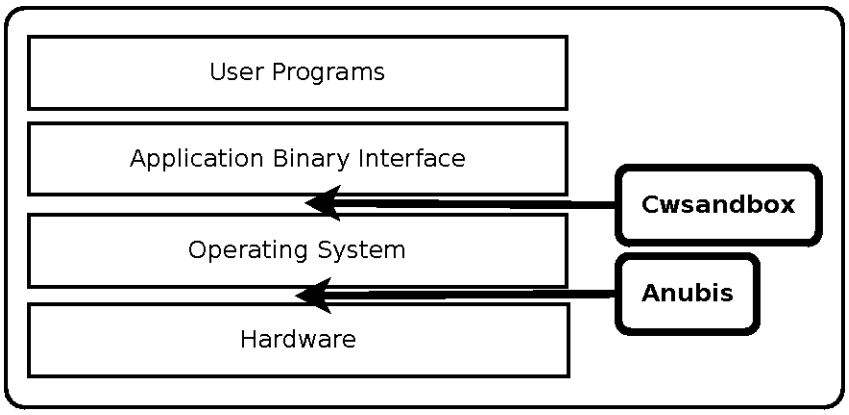

Figure 4.1. The different levels Anubis and Cwsandbox operate on

4.1.2. Classifiers

Experimental evaluation using a classifiers built with a single learning algorithm may provide useful insights, but there is a risk that any results found are due to an unknown bias present in the algorithm. Thus, an optimal experimental evaluation would involve creating classifiers using a randomly selected, representative sampling of learning algorithms from the universe of all possible learning algorithms. However, this simply isn't feasible. Instead, to minimize the risk of a bias, a small number of well known learning algorithms were used. While a single learning algorithm may favor a specific type of feature set, it is not likely that different learning algorithms will share the same bias.

The learning algorithms chosen were: Naïve Bayes, Rule Induction, Decision Tree, and K-Nearest Neighbor (KNN) as implemented by the `RapidMiner` application. Other learning algorithms available in `RapidMiner` were not included in the evaluation because they were very similar to the classifiers already listed, and initial evaluations

`http://rapid-i.com`.

indicated that including them would not add any significant value. Specifically, Naïve Bayes with kernel density estimation, Decision Stump, Random Tree, Random Forest, and Single Rule Induction classifiers were found to have similar performance patterns to the classifiers presented.

Naïve Bayes A classifier built using the Naïve Bayes learning algorithm performs classification based on a theory of probability known as Bayes' theorem [18]. Bayes' theorem states:

$$P(c_k|x) = \frac{P(c_k)P(x|c_k)}{P(x)} \tag{4.1.1}$$

Where $C = \{c_1, ..., c_k, ..., c_n\}$ is the set of all possible classes and $x = \{f_1, ..., f_m\}$ is a given feature set. Intuitively, Bayes theorem states that the probability of a sample with feature set x being a member of class k is the same as the probability of any sample belonging to class k, times the probability that a member of class k has feature set x, divided by the probability of any sample having feature set x. Thus, if we know $P(c_k|x)$ for all c_k in C, then the process of assigning a sample with feature set x to a class is as simple as finding the class which has the highest value for $P(c_k|x)$.

While $P(c_k)$ and $P(x)$ can be directly computed from the data, $P(x|c_k)$ poses a challenge, as the number of possible values of x grows exponentially with the number of features in x. However, if we assume that all the features in x are statistically independent of one another, we can model the conditional probability as follows:

$$P(x|c_k) = \prod_{i=1}^{m} P(x_i|c_k) \tag{4.1.2}$$

In other words, if independence is assumed, the conditional probability of the feature set can be modeled by the product of the conditional probabilities of the individual values of the set.

Thus the class of a sample having feature set x can be computed by finding the value of $c \in C$ which maximizes the equation:

$$P(c|x) = \frac{P(c)\prod_{i=1}^{m} P(x_i|c)}{P(x)} \tag{4.1.3}$$

Rule Induction The Rule Induction learning algorithm operates by building a set of rules of the form: **if** <*Conditions*> **then** *Class*. The <*Conditions*> are typically a conjunction of *Feature* = *Value* tests. A sample is classified by applying the rules to its feature set. The class of the sample is given by the rules whose conditions are satisfied.

While there is a collection of algorithms for building the rules [19, 20, 21], the algorithm implemented by `RapidMiner` is based off RIPPER [22]. Given a list of the classes, $C_1, C_2, ..., C_n$, ordered so that the least common class is first and the most common class is last, beginning with C_1 and going in order, a rule is iteratively built for each class until all samples are covered by a rule or only class C_k remains. A rule is built by greedily adding the condition with the maximum information gain to the rule until the rule covers all samples in the class or the error rate of the rule exceeds 50%. A sample is said to be covered by a rule if it satisfies the conditions of the rule. After each rule is built, it is immediately pruned to increase the accuracy of the rule. A rule is pruned by deleting the combination of conditions which maximizes the function $\frac{p}{p+n}$ where p is the number of samples covered by the rule in the class (true positives) and n is the number of samples covered by the rule not in the class (false positives).

Decision Tree The Decision Tree learning algorithm is similar in concept to Rule Induction, but instead of building a set of self contained rules, a tree is built where the nodes of the tree represent the features, the edges correspond to the values which the feature can take, and the leafs give the different classes. A sample is classified by

starting at the root of the tree and following the edges that match the values of the sample's features until a leaf node is reached. This class represented by the leaf node is the class given to the sample. A very simple Decision Tree to determine if a fruit is an apple, orange, or banana is given in Figure 4.2.

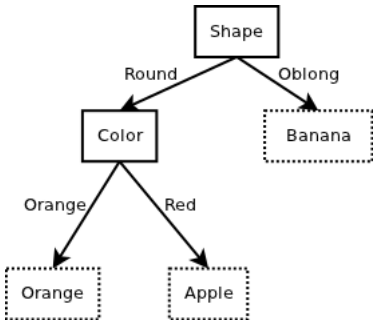

Figure 4.2. A simple Decision Tree

The algorithm used by `RapidMiner` to generate the Decision Tree is ID3 [23]. Given the list of features, $f_1, f_2, ..., f_n$, ID3 chooses the root of the tree to be the feature with the highest information gain and creates a branch for every value this feature can have. ID3 then repeats this process for each of the newly created branches, choosing the feature represented by the new node to be the feature that has not been selected on that path with the highest information gain. Leaves are created when either all samples whose features have the values specified by current path are of the same class or adding the next feature fails to obtain a specified minimum improvement.

K-Nearest Neighbor The K-Nearest Neighbor learning algorithm builds a type of classifier known as a "lazy classifier." Unlike the other classifiers presented, all computations are "procrastinated" until the classification is performed. K-Nearest Neighbor starts with a set of samples whose features and classes are known to the classifier. When an unknown sample is classified, distance scores are computed between the unknown sample and all known samples. These scores are a measure of how similar

(or dissimilar) one sample is to another, and can be computed a number of different ways. The unknown sample is then assigned to the most common class among the k samples which are most similar to the unknown sample.

The distance function used by `RapidMiner` is a simple edit distance, or the number of changes that would need to be made in order to make the two samples identical. To compute this distance score, the two feature sets of the samples are compared, and for every feature that is present in one set and not the other, the distance score is incremented. Using edit distance, the distance score can range from zero to the number of total possible features. The lower the distance score for two samples, the more similar they are, with zero indicating the samples are identical.

4.1.3. Fusers

Feature fusion was implemented by a set of python scripts. These scripts parsed the outputs of Anubis and Cwsandbox, extracted the features, performed the fusion algorithm, and output the feature sets in a format the classifiers could understand.

4.2. Data Set

4.2.1. Malware Samples

The set of malware used was composed of six families plus benign assembled from three different sources. The samples belonging to the Agent, Parite, PcClient, and the Benign families were obtained from a commercial anti-virus vendor. The Banker and SDBot families were provided by OffensiveComputing.net. Finally, the storm class was self collected from the storm botnet. In order to establish the validity of the class label assigned to the samples, each sample was tested by ensuring that at least two of three current Anti-Virus products listed the malware as belonging to the family. The benign samples were collected from system and third party executables from clean Windows

2000 and Windows Vista systems. Table 4.1 gives the number of malware samples in each family.

The originally assembled collection of malware contained 20,000 samples, but only 18,422 samples were successfully analyzed by the sensors chosen. A sensor was considered to have successfully analyzed a malware file if the file executed. Even if there were no features detected, as long as the program being analyzed was able to load and begin execution, the analysis was still considered successful. Only the 18,422 samples which were successfully analyzed were used in the classifications because fusion by disjoint union is undefined otherwise.

Family	**# Samples**	**% of Collection**	**Source**
sdbot	5,956	32.33%	OffensiveComputing.net
agent	1,930	10.48%	Commercial AV
benign	5,460	29.64%	Commercial AV
banker	1,118	6.07%	OffensiveComputing.net
pcclient	1,562	8.49%	Commercial AV
parite	1,841	9.99%	Commercial AV
storm	575	3.12%	Self Collected

Table 4.1. Malware samples per family

4.2.2. Features

The features used for classification are an adaptation of ideas presented in [17] and [24]. A feature is an action taken on a resource, divided into three parts: category, action, and data. The category of a feature corresponds to the type of object that an action was taken on. These are things such as DLLs, files, processes, etc. The action of a feature is, as the name implies, the action taken on the object, such as load or create. The data part of a feature is some data which uniquely identifies the object being acted upon. For example, the data stored for files is the path to the file and the data stored

Category	Action	Data
DLL	load	path
Mutex	create	name
Registry	create modify read	key
File	create/modify read	path
Process	create	name

Table 4.2. List of possible features

for registry actions is the registry key. The list of possible features are given in Table 4.2 with concrete examples given in Table 4.3.

Category	Action	Data
DLL	load	`c:\windows\system32\mlang.dll`
Mutex	create	aashea
Registry	read	`hklm\software\classes\chkfile`
File	read	`c:\windows\winvrn.exe`
Process	create	`c:\windows\charmapnt.exe`

Table 4.3. Examples of extracted features

4.3. Procedure

The overall procedure followed was to submit the malware samples to the sensors and retrieve the raw data, extract the features from the raw data, create the fused and non-fused feature sets, determine classification accuracies, and perform statistical analysis.

The feature set reductions, cross validation, and the classifications for the paired t-test and confidence intervals were performed using the open source tool RapidMiner

[25]. The dependent pairs t-tests and the confidence intervals were computed using the Data Analysis ToolPak in Microsoft Excel (2007).

4.3.1. Data Generation

The collection of malware was first submitted to both Anubis and Cwsandbox and the outputs retrieved. From these outputs, the feature sets were then extracted. The feature sets were then reduced using information gain and the fused feature sets created. The reduced feature sets were then used to perform K-Fold Cross Validation and Hold-Out Classification. The accuracies were generated using the 10-Fold Cross Validation and the statistical tests were performed on the data from the Hold-Out Classification.

There were four types of feature sets extracted. The Anubis feature set contained only features that were detected by Anubis, the Cwsandbox feature set contained only features detected by Cwsandbox, the Union feature set was created using union as the fusion function, and the Disjoint feature set was created using disjoint union as the fusion function. There were a total of 58,924 features detected by either Anubis or Cwsandbox, 6,697 features were detected by both Anubis and Cwsandbox, 18,070 features were unique to Anubis, and 34,157 features were unique to Cwsandbox. The total number of features for each feature type are given in Table 4.4.

Feature Type	**Before Reduction**	**After Reduction**
Anubis	24,767	661
Cwsandbox	40,854	906
Union	58,924	1,363
Disjoint Union	58,924	1,265

Table 4.4. Number of features before and after reduction

4.3.2. Accuracy Generation

10-Fold Cross Validation was used to obtain accuracy estimations for all combinations of classifier and feature set types. To perform 10-Fold cross validation, the sample data is divided into 10 sets, called folds, of equal size. One of these ten folds is chosen as the testing set, and the remaining folds comprise the training set. The classifier is then trained and tested using these sets and the number of correct and incorrect classifications recorded. This process is repeated ten times, each time choosing a previously unchosen fold as the test set, so that each fold is used as the test set exactly once. The overall accuracy of the classifier is then determined by dividing the number of correct classifications by the total number of classifications done.

4.3.3. Statistical Tests

The two statistical tests used were paired t-tests and 95% confidence intervals. These tests were only performed against combinations of feature sets using the same classifier. The experiment was only focused on how the feature sets used affected classification accuracy, not how the classifier itself affected accuracy. In the same way, statistical tests were not performed to compare classifications using Cwsandbox feature set against those using the Anubis feature set.

The two statistical tests chosen cannot be performed using the data from the Cross Validation classifications. The reason is that both the paired t-test and the confidence interval assume a pairing of data between the classifications being compared. In other words, the training set (set of labeled data the learning algorithm takes as input) and the test set (the set of data used to test the classifier created by the learning algorithm) need to consist of the same files between the classifications being compared. In addition, the result of each individual classification needs to be recorded. The

classification tool being used (`RapidMiner`) does not provide a mechanism to ensure the needed pairing or output the results of individual classifications for Cross Validation.

To generate the data needed for statistical analysis, Hold-Out classification was performed and the result of individual classifications recorded along with any differences in these individual classifications. Hold-Out Classification works by simply "holding out" a subset of the data set to be used as the testing set and the rest is used as the training set. Unlike Cross Validation, there is no rotating of training and test sets. The initial training and test sets selected are the only sets used. In order to ensure the necessary pairing, a single training and test set were used for all classifications. When recording the results of individual classifications, a correct classification was recorded as a 1 and an incorrect classification was recorded as a 0. The differences in individual classifications were recorded as the arithmetic difference. In other words, a zero meant the classifications were the same, a 1 indicated that the first classification was correct while the second was not, and a -1 indicated that the first classification was incorrect while the second one was correct. Table 4.5 gives an example of these recordings for a comparison between classifications using Disjoint and Union feature types.

A paired t-test is used in order to determine if the difference between accuracies observed using different feature types is a statistically significant difference. To perform a paired t-test, Hold-Out Classification is performed and the results recorded as described above and illustrated in Figure 4.5. Once these recordings are obtained, a t-score, t_d, is calculated using the formula $t_d = \dfrac{\bar{d}}{SE}$ where $\bar{d}$ is the average of the differences and SE is the standard error of the differences. SE is given by $\dfrac{\sigma_{\bar{d}}}{\sqrt{n}}$ where $\sigma_{\bar{d}}$ is the standard deviation of the differences and n is the size of the test set. The p-value for this t-score is then computed. The p-value is the probability that the t-score is the result of chance, given there is no statistical significance. In other words, if the classifications were to be repeated, the p-value gives the probability that the same

difference in accuracies would be observed. If this probability is sufficiently low, specifically if it less than 0.05, then the difference in accuracies is said to be statistically significant. The mathematical methods used to compute the p-value are outside the scope of this thesis. Tables exist online which list estimates of p-values for given t-scores and many statistical software packages can provide exact p-values.

Sample	**Disjoint**	**Union**	**Difference**
banker.32489c8	1	1	0
sdbot.fdb1ded2	1	0	1
parite.499ab0f	0	1	-1
sdbot.da572729	1	1	0
benign.f0037b3	0	1	-1
sdbot.6234aaec	1	0	1
benign.9cf2d8f	1	0	1
sdbot.533e9f9f	0	0	0
banker.a1a5c20	0	1	-1

Table 4.5. Example of classification results used for statistical tests

In addition to the paired t-tests, 95% confidence intervals are also computed as a statistical test. A 95% confidence interval is an interval in which the difference between accuracies has a 95% probability of being in. In other words, given a 95% confidence interval for two classifications, the difference in accuracies obtained if the classifications were repeated with any training and test sets has a 95% chance of being between the upper and lower bounds of the given interval. To compute a 95% confidence interval, Hold-Out classification is performed and the data recorded as described above and illustrated in Figure 4.5. The confidence interval is then computed by $\bar{d} \pm 1.96 t_d$. Where $\bar{d}$ and t_d are defined as they were for the paired t-test. If this interval does not cross 0 (the upper and lower bounds have the same sign), then statistical significance is achieved.

4.4. Results

The overall accuracies obtained from Cross Validation are given in Table 4.6. The columns list the classifiers used and the rows list the feature types. The bold accuracies indicate the feature type with the highest accuracy for each classifier. For the Naïve Bayes classifier, the Cwsandbox feature set achieved the highest accuracy with 81.40%. For the Rule Induction, Decision Tree, and KNN classifiers, Disjoint Union had the highest accuracies with 91.30%, 92.83%, and 95.30% respectively.

Feature Type	**Naïve Bayes**	**Rule Induction**	**Decision Tree**	**KNN**
Anubis	67.78%	75.32%	78.20%	86.29%
Cwsandbox	**81.40%**	86.98%	88.91%	91.65%
Union	76.45%	84.11%	90.48%	92.85%
Disjoint	79.88%	**91.30%**	**92.83%**	**95.30%**

Table 4.6. Classification accuracies

The p-values from the paired t-tests are given in Table 4.7. The columns give the classifiers and the rows give the feature types. The Union & Anubis row, for example, gives the p-values calculated when comparing the Union and Anubis feature sets. The difference in accuracies between the two feature sets is statistically significant if the p-value is less than 0.05. The places where statistical significance is not achieved are indicated by bold text. This is only between the Union and Cwsandbox feature sets for the KNN classifier (6.11×10^{-2}), the Disjoint and Cwsandbox feature sets using the Naïve Bayes classifier (2.02×10^{-1}), and the Disjoint and Union feature sets using the Decision Tree classifier (8.41×10^{-2}).

The confidence intervals are given in Table 4.8. The columns give the classifiers and the rows give the feature types being compared. For example, the Disjoint - Union row gives the 95% confidence intervals for the difference in accuracies between the Disjoint and Union feature sets. Statistical significance is reached if this interval does not cross

Feature Types	Naïve Bayes	Rule Induction	Decision Tree	KNN
Union & Anubis	1.00E-16	1.00E-16	1.00E-16	1.00E-16
Disjoint & Anubis	1.00E-16	1.00E-16	1.00E-16	1.00E-16
Union & Cwsandbox	2.29E-6	1.11E-4	1.33E-4	**6.11E-2**
Disjoint & Cwsandbox	**2.02E-1**	9.42E-7	8.58E-9	1.63E-9
Disjoint & Union	1.00E-15	1.00E-13	**8.41E-2**	1.02E-6

Table 4.7. P-values from the paired t-tests

zero. The places were statistical significance is not reached are indicated by bold text. The confidence intervals further validate the results of the p-values as statistical significance is indicated by the confidence interval if and only if the p-value indicated statistical significance.

Feature Comparison		Naïve Bayes	Rule Induction	Decision Tree	KNN
Union - Anubis	**min**	0.0788	0.0818	0.112	0.0933
	max	0.106	0.119	0.145	0.123
Disjoint - Anubis	**min**	0.116	0.139	0.117	0.112
	max	0.149	0.178	0.154	0.145
Union - Cwsandbox	**min**	-0.0695	-0.0477	0.0111	**-0.0025**
	max	-0.0279	-0.0147	0.0370	**0.0211**
Disjoint - Cwsandbox	**min**	**-0.0293**	0.0155	0.0204	0.0198
	max	**0.0118**	0.0370	0.0420	0.0393
Disjoint - Union	**min**	0.0302	0.0422	**-0.00301**	0.0119
	max	0.0497	0.0727	**0.01720**	0.0286

Table 4.8. 95% Confidence Intervals

4.5. Discussion

The results support the claim that disjoint union improves the accuracy of malware classification. As can be seen in Table 4.6, disjoint union typically obtained a statistically significant higher accuracy than the classifications performed without feature fusion. There is only one example when there was no increase. For the Naïve Bayes classifier, the Cwsandbox feature set had an accuracy of 80.40%, while disjoint

union only had an accuracy of 79.88%. However, according to Table 4.7, the p-value for these two classifications is 0.202. Since this value is greater than 0.05, statistical significance is not achieved. All increases in accuracy disjoint union obtained over one of the non-fused feature sets, however, were statistically significant. This can be seen in both Table 4.6 and Table 4.7. For the Rule Induction, Decision Tree, and KNN classifiers, the Cwsandbox feature set obtained accuracies of 86.98%, 88.91%, and 91.65%, while disjoint union obtained higher accuracies of 91.30%, 92.83%, 95.30%. The p-values for these classifications were 9.42E-7, 8.58E-9, and 1.63E-9.

The results also lend evidence to support the claim that the reason disjoint union improves malware classification is because it enables the classifiers to take advantage of the patterns of "missing" features created by obfuscations. This can be seen by comparing the accuracies of Disjoint features to Union features. If the classifications were not taking advantage of the patterns recorded through disjoint union, then the performance between Disjoint and Union should be comparable. However, looking at Table 4.6, we find that Disjoint consistently achieves a higher accuracy than Union (79.88% vs. 76.45%, 91.30% vs. 84.11%, 92.83% vs. 90.48%, 95.30% vs. 92.85%). Looking at Table 4.7, we see that these increases are statistically significant, except when using the Decision Tree classifier. For the Naïve Bayes, Rule Induction, and KNN classifiers, the p-values were 1E-15, 1E-13, and 1.02E-6, respectively. For the Decision Tree classifier, the p-value was 8.41E-2. Lending further evidence is the fact that while disjoint union consistently achieved higher accuracies than no fusion, Union only performed better than the Cwsandbox feature set when using the Decision Tree and KNN classifiers (90.48% vs. 88.91% and 92.85% vs. 91.65%), and only the increase using the Decision Tree classifier was statistically significant with a p-value of 1.33E-4 (KNN had p-value of 6.11E-2). Thus, based upon this evidence, can be concluded that feature fusion by disjoint union is indeed giving additional information to the classifier.

Since the only difference between union and disjoint union is recording which sensor recorded which feature, it can be reasonably surmised that the additional information provided contains the obfuscation patterns.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

This thesis has presented a novel method for feature fusion across domains using disjoint union in order to capitalize on the fact that the obfuscations used by malware do not defeat all sensors equally. This method was evaluated using a case study and the results were presented. It was found that with the sensors and classifiers chosen, feature fusion by disjoint union does typically increase the accuracy of classification by a statistically significant amount. In addition, feature fusion by disjoint union was found to perform better than feature fusion by union when fusing across domains, giving evidence to the claim that feature fusion by disjoint union generates obfuscation patterns which are used by the classifier.

There are a number of open questions and areas of research not addressed by this thesis. While the experimental results imply that fusion by disjoint union provides an additional layer of information to the classifier, it is not clear what this data is exactly. One area of further research would be to investigate this additional information to determine if they are indeed obfuscation patterns and to develop theoretical models to describe this additional information. If a theoretical model for obfuscation patterns is created, it could be applied to other areas of malware analysis, such as obfuscation detection.

Another area that warrants further study is the fusion function. Disjoin union is a simple fusion function and a more complex function may be able to provide even more information to the classifier. For example, a fusion function which takes into consideration the reliability of the sensors may be useful.

Finally, this thesis applied a single type of information fusion to a single type of malware analysis. Applications of information fusion to malware analysis is still under explored. Future research can be done in applying the different types and methods of information fusion to different types of malware analysis.

BIBLIOGRAPHY

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *Advances in Cryptology – CRYPTO 2001: Proceedings of the 21st Annual International Cryptology Conference*, J. Killian, Ed., 2001. [Online]. Available: http://www.wisdom.weizmann.ac.il/~oded/p_obfuscate.html

[2] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *Proceedings of the 13th USENIX Security Symposium*. Usenix, 2004, pp. 255–270.

[3] Y. Lu, S. Din, C. Zheng, and B. Gao, "Using multi-feature and classifier ensembles to improve malware detection," *Journal of C.C.I.T.*, vol. 39, no. 2, November 2010.

[4] M. Masud, L. Khan, and B. Thuraisingham, "A scalable multi-level feature extraction technique to detect malicious executables," *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, 2008.

[5] R. Islam, R. Tian, L. Batten, and S. Versteeg, "Classification of malware based on string and function feature selection," in *Second Cybercrime and Trustworthy Computing Workshop*. IEEE Computer Society, 2010, pp. 9–17.

[6] D. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[7] R. Boudjemaa and A. Forbes, "Parameter estimation methods for data fusion," *NPL Report CMSC*, vol. 38, no. 04, 2004.

[8] E. Nakamura, A. Loureiro, and A. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Computing Surveys (CSUR)*, vol. 39, no. 3, p. 9, 2007.

[9] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFRS," *Philosophical Transactions*, vol. 53, p. 370, 1763.

[10] M. Masud, L. Khan, and B. Thuraisingham, "A hybrid model to detect malicious executables," in *Proc. of the IEEE International Conference on Communication*, 2007, pp. 1443–1448.

[11] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation," *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 500–509, 2010.

[12] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483–1494, 2009.

[13] E. Menahem, L. Rokach, and Y. Elovici, "Troika-an improved stacking schema for classification tasks," *Information Sciences*, vol. 179, no. 24, pp. 4097–4122, 2009.

[14] (2011, June) Anubis: Analyzing unknown binaries. [Online]. Available: http://anubis.iseclab.org

[15] (2011, June) Cwsandbox: Behavior-based malware analysis. [Online]. Available: http://mwanalysis.org

[16] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *Security & Privacy, IEEE*, vol. 5, no. 2, pp. 32–39, 2007.

[17] U. Bayer and C. Kruegel, "Ttanalyze: A tool for analyzing malware," *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 2006.

[18] D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," *Machine Learning: ECML-98*, pp. 4–15, 1998.

[19] J. Grzymala-Busse, "A new version of the rule induction system LERS," *Fundamenta Informaticae*, vol. 31, no. 1, pp. 27–39, 1997.

[20] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *Machine learning—EWSL-91*. Springer, 1991, pp. 151–163.

[21] C. Nevill-Manning, G. Holmes, and I. Witten, "The development of Holte's 1R classifier," in *Proc. Conf. on Artificial Neural Networks and Expert Systems*, Nov. 1995, pp. 239–242.

[22] W. Cohen, "Fast effective rule induction," in *Machine Learning-International Workshop Then Conference*. Morgan Kaufmann Publishers, INC., 1995, pp. 115–123.

[23] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[24] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," 2009. [Online]. Available: https://mwanalysis.org/inmas/maschinellesLernen/mist/mist-tr.pdf

[25] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler, "Yale: Rapid prototyping for complex data mining tasks," in *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, Eds. New York, NY, USA: Acm, 2006, pp. 935–940.

LeDoux, Charles. Bachelor of Science, University of Louisiana at Lafayette, Spring 2005; Master of Science, University of Louisiana at Lafayette, Fall 2011
Major: Computer Science
Title of Thesis: Feature Fusion Across Domains for Improved Malware Classification
Thesis Director: Dr. Arun Lakhotia
Pages in Thesis: 48; Words in Abstract: 175

ABSTRACT

Malware authors often use obfuscations to defeat the "sensors" used by malware analysts. Since universal obfuscation is infeasible, the deception techniques used by malware authors are necessarily targeted towards specific sensors. This thesis presents a method of feature fusion across domains to combine similar features observed by multiple sensors operating over different domains in order to improve classifier accuracy. The key innovation lies both in the method of fusion and in the choice of sensors. Sensors are chosen which observe the same features but use different mechanisms such that they do not share the same weaknesses. These features are fused using disjoint union. It is hypothesized that the patterns of "missing" features between the sensors provides additional information related to the obfuscations used and so will improve classification accuracy. Experimental results, on a dataset of over 18,000 samples, show that while a simple union of the features collected from sensors on different domains only slightly increases the accuracy of classification, fusing these features using disjoint union generally increases classification accuracy by a statistically significant amount.

BIOGRAPHICAL SKETCH

Charles LeDoux was born on December 12, 1987 in Eunice, Louisiana to Joseph and Karen LeDoux. He entered the University of Louisiana at Lafayette in the fall of 2005 where he received his Bachelor of Science in Computer Science in the spring of 2009. Immediately after receiving his Bachelor Degree, Charles was accepted into the graduate program at the University of Louisiana at Lafayette. He is to receive his Master of Science in Computer Science in the fall of 2011.