CLASSIFICATION OF POTENTIALLY UNWANTED PROGRAMS USING SUPERVISED LEARNING

Raja Muhammad Khurram Shahzad

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2013:02

School of Computing

Classification of Potentially Unwanted Programs Using Supervised Learning

Raja Muhammad Khurram Shahzad

Blekinge Institute of Technology licentiate dissertation series
No 2013:02

Classification of Potentially Unwanted Programs Using Supervised Learning

Raja Muhammad Khurram Shahzad

Licentiate Dissertation in
Computer Science

School of Computing
Blekinge Institute of Technology
SWEDEN

2013 Raja Muhammad Khurram Shahzad
School of Computing
Publisher: Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Printed by Printfabriken, Karlskrona, Sweden 2013
ISBN: 978-91-7295-247-8
ISSN 1650-2140
urn:nbn:se:bth-00548

Abstract

Malicious software authors have shifted their focus from illegal and clearly malicious software to potentially unwanted programs (PUPs) to earn revenue. PUPs blur the border between legitimate and illegitimate programs, and thus fall within a grey zone. Existing anti-virus and anti-spyware software are, in many instances, unable to detect previously unseen or zero-day attacks and separate PUPs from legitimate software. Many tools also require frequent updates to be effective. By predicting the class of a particular piece of software, users can get support before taking the decision to install the software. This Licentiate thesis introduces approaches to distinguish PUP from legitimate software based on the supervised learning of file features represented as *n*-grams.

The overall research method applied in this thesis is experiments. For these experiments, malicious software applications were obtained from anti-malware industrial partners. The legitimate software applications were collected from various online repositories. The general steps of supervised learning, from data preparation (*n*-gram generation) to evaluation, were followed. Different data representations, such as byte codes and operation codes, with different configurations, such as fixed-size, variable-length, and overlap, were investigated to generate different *n*-gram sizes. The experimental variables were controlled to measure the correlation between *n*-gram size, the number of features required for optimal training, and classifier performance.

The thesis results suggest that, despite the subtle difference between legitimate software and PUP, this type of software can be classified accurately with a low false positive and false negative rate. The thesis results further suggest an optimal size of operation code-based *n*-gram for data representation. Finally, the results indicate that classification accuracy can be increased by using a customized ensemble learner who makes use of multiple representations of the data set. The investigated approaches can be implemented as a software tool with a less frequently required update in comparison to existing commercial tools.

Acknowledgments

I owe a great debt of gratitude and thanks to my supervisor, Dr. Niklas Lavesson for his trust and triggering my interest in Machine Learning. Throughout my research work, Niklas has been a great mentor and a source of encouragement. He has provided advice, guidance, and taught me innumerable things from the basic research to develop new ideas. He has inculcated me how to approach each research problem with a different angle and ascertain the validity of the applied approach. Thank you Niklas, I have still to learn a lot from you.

This thesis would not have happened without the invaluable support, inspiration, and guidance of my advisor, Professor Bengt Carlsson. His ideas and feedback have been instrumental in steering this research and improving the quality of work. I am heartily thankful to Dr. Martin Boldt for his guidance, support, and discussion on the research and as well as photography. Furthermore, I like to thank my colleagues and fellow Ph.D. students at DISL.

I place on record my sense of gratitude to Higher Education Commission (HEC), Pakistan and Blekinge Institute of Technology, Sweden for providing me the opportunity to pursue the graduate studies. I like to extend my thanks to Conny Johansson, Dean of School of Computing, for having trust about my stay at BTH, for having welcomed me as a research student and providing all the necessary facilities. Monica H. Nilsson, Kent Adolfsson, Eva-Lotta Runesson, and Karin Svahnberg I love your administrative help and thank you for being my supporter.

Preface

This thesis presents the work done in the following four papers. These four papers are published in peer-reviewed conference proceedings and a journal. The thesis author is the main author of all the papers. The included versions of Paper I, Paper II, and Paper III contain minor modifications/expansions in the text in comparison to the original publications. The modifications are to correct the minor mistakes found in the original publications and update the references such as updating the status of a publication from submitted to the published information.

Paper I. Raja Khurram Shahzad, Syed Imran Haider, and Niklas Lavesson, "Detection of Spyware by Mining Executable Files," in the *Proceedings of Fifth International Conference on Availability, Reliability, and Security*, IEEE Press, 2010.

Paper II. Raja Khurram Shahzad and Niklas Lavesson, "Detecting Scareware by Mining Variable Length Instruction Sequences," in the *Proceedings of Tenth Annual Information Security South Africa Conference*, IEEE Press, 2011.

Paper III. Raja Khurram Shahzad, Niklas Lavesson, and Henric Johnson, "Accurate Adware Detection using Opcode Sequence Extraction," in the *Proceedings of Sixth International Conference on Availability, Reliability, and Security*, IEEE Press, 2011.

Paper IV. Raja Khurram Shahzad, Niklas Lavesson, "Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection", in press, *Speical Issue of Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, Vol. 4, No. 1, 2013.

As a main author, the thesis author was responsible for conducting the research study, performing experiments, and writing papers. The Paper I was the extension of author's master thesis. The co-authors of the Paper I were the thesis partner and the thesis supervisor. The co-authors of the Paper II, Paper III, and Paper IV were supervisor and senior researchers in the Distributed and Intelligent Systems Laboratory, who aided in research study design, analysis, and writing papers by providing their iterative feedbacks.

The following articles are excluded from the thesis, but are related to the thesis scope.

Paper V. Raja Khurram Shahzad and Niklas Lavesson, "Detecting Scareware by Mining Variable Length Instruction Sequences," Extended abstract, in the *Proceedings of 11th Scandinavian Conference on Artificial Intelligence*, IOS Press, 2011.

Paper VI. Raja Khurram Shahzad and Niklas Lavesson, "Veto-based Malware Detection", in the *Proceedings of Seventh International Conference on Availability, Reliability, and Security*, IEEE Press, 2011.

Contents

Acknowledgments **iii**

Preface **v**

Contents **vii**

1 Introduction **1**

2 Background **5**
2.1 Malicious Software 5
2.2 Malware Detection 7
2.3 Terminology 10
2.4 Related Work 13

3 Primary Research Contributions **19**
3.1 Aim and Scope 20
3.2 Contributions 20

4 Research Approach **25**
4.1 Research Questions 26
4.2 Research Methodology 30
4.3 Research Validity 32

5 Conclusions **35**
5.1 Discussion 35

5.2 Conclusion 37
5.3 Future Work 39

6 Detection of Spyware by Mining Executable Files 41
Raja Khurram Shahzad, Syed Imran Haider, Niklas Lavesson
6.1 Introduction 42
6.2 Background 44
6.3 Proposed Method 49
6.3.1 Data Collection 49
6.3.2 Malicious File Percentage 50
6.3.3 Byte Sequence Generation 50
6.3.4 *n*-gram Size 50
6.3.5 Parsing 50
6.3.6 Feature Extraction 50
6.3.7 Feature Reduction 51
6.3.8 ARFF Generation 52
6.3.9 Classifiers 53
6.3.10 Performance Evaluation Criteria 54
6.4 Results 55
6.4.1 Results for $n = 4$ 55
6.4.2 Results for $n = 5$ 57
6.4.3 Results for $n = 6$ 58
6.5 Discussion 58
6.6 Conclusions and Future Work 60

7 Detecting Scareware by Mining Variable Length Instruction Sequences 63
Raja Khurram Shahzad, Niklas Lavesson
7.1 Introduction 64
7.1.1 Background 65
7.1.2 Traditional Countermeasures 67
7.1.3 Scope and Aim 68
7.1.4 Outline 68
7.2 Related Work 69
7.2.1 Concepts and Terminology 69

7.2.2 Related Directions of Research 71
7.3 Methodology 73
7.3.1 File Sampling and Data Set Design 73
7.3.2 Extraction and Data Preparation 74
7.3.3 Feature Selection 75
7.4 Experiment 76
7.4.1 Learning algorithms 76
7.4.2 Evaluation 78
7.5 Results 79
7.6 Analysis 81
7.7 Conclusions and Future work 82

8 Accurate Adware Detection using Opcode Sequence Extraction 85
Raja Khurram Shahzad, Niklas Lavesson, Henric Johnson
8.1 Introduction 86
8.1.1 Aim and Scope 87
8.2 Background 88
8.2.1 Data Mining-based Detection 88
8.2.2 Feature Selection 89
8.3 Related Work 90
8.4 Method 91
8.4.1 Overview 92
8.4.2 Data Set Generation 92
8.4.3 File Size and Data Size Analysis 93
8.4.4 Disassembly and Opcode Extraction 93
8.4.5 Parsing and *n*-Gram Size 94
8.4.6 Feature Selection 94
8.4.7 Data mining algorithms 96
8.5 Evaluation Metrics 99
8.6 Experimental Procedure 100
8.7 Results 100
8.8 Analysis 101
8.8.1 Algorithm Performance Analysis 102
8.8.2 State-of-the-Art 103
8.8.3 Opcode Analysis 103

8.8.4 Practical Considerations 104
8.9 Conclusion and Future work 104

9 Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection 107
Raja Khurram Shahzad, Niklas Lavesson
9.1 Introduction 108
9.1.1 Aim and Scope 110
9.1.2 Contribution 110
9.1.3 Outline 110
9.2 Background 111
9.2.1 Terminology 112
9.2.2 Related Work 113
9.3 Veto-based Classification 115
9.3.1 Voting Rules 116
9.3.2 Architecture 118
9.4 Pre-processing Techniques 118
9.4.1 Overlapping *n*-grams 120
9.4.2 Non-adjacent Opcode Extraction 120
9.5 Experiment I 121
9.5.1 Feature Representation 123
9.5.2 Data Set Creation 123
9.5.3 Pre-Processing 124
9.5.4 Feature Selection 125
9.5.5 Performance Evaluation Criteria 126
9.5.6 Pre-Experiment for Algorithm Selection 126
9.5.7 Results and Discussion 127
9.6 Experiment II 130
9.6.1 Trust-based Veto algorithm 131
9.6.2 Results and Discussion 135
9.7 Conclusion and Future Work 137

Bibliography 139

One

Introduction

Networked communication technologies are trusted for performing various routine operations to transfer sensitive information such as financial transactions. One of the challenges of these technologies is the manifestation of malicious software (malware), which is used to disrupt communication and damage data [1]. In the last decade; the landscape of malware has changed significantly from traditional malware (i.e., viruses, worms, and Trojan horses) to programs having unwanted functionalities such as users' sensitive information collection, e.g., personal information, credit card information, and banking information [2]. This type of software may obtain an uninformed consent of the user, and is, therefore, referred to as potentially unwanted programs (PUPs) [3,4,5,6]. There is no consensus on the precise definition of PUP; however, in this thesis, the term is defined as follows:

A piece of software that deceives the user, mainly for commercial purposes, and that may negatively affects the user's security and/or privacy.

PUPs mainly constitute a threat against privacy; that is a user's right to determine the extent of sharing the sensitive information [7]. PUP vendors make different types of PUPs, i.e., spyware, adware, and scareware [8]. The main objective of these types is to earn revenue by invading privacy. The revenue generated from PUPs is increasing constantly due to the high return over investment [9]. It is reported that one in seven security breaches involves PUP [3]. In addition to the revenue generation, another explanation for this increase is that different types of PUPs are declared legal by current statutes, if the vendor obtains the consent of the user during the program installation [10]. This fact has encouraged the PUP vendors to launch malicious code at an increasing rate.

Traditional malware is primarily addressed through a reactive approach, i.e., malware detection at the user's computer. Malware are detected using either pattern matching, i.e., signature-based detection, or heuristic-based detection. Signature-based approach is incapable of detecting zero-day attacks and is only capable of detecting circa 30 % of instances in the wild [11]. Heuristic-based approach produce high false alarms and is capable of detection circa 39 % of instances in the wild [12]. It is also difficult to establish an acceptable borderline between the PUPs and legitimate software for the heuristic-based approach [12]. Signature-based approaches or heuristic based approaches are also applied for the PUP detection. There is an arm race, between PUP vendors and anti-malware vendors, to circumvent the traditional detection approaches. A PUP vendor may challenge the anti-PUP software classification on the grounds that anti-PUP software lack accuracy and fairness, i.e., biased towards a class of applications [13]. Another problem associated with the PUPs classification is that there is no consensus about the legal status of PUPs as they reside in the grey area between legitimate (or benign) software and malware [10].

In response to the increasing number of PUPs in wild, effective and automatic PUPs detection/classification and analysis techniques are required. Therefore, the focus of this dissertation is to extend the heuristic-based detection method by using the supervised learning for the automated classification and analysis of PUPs. Supervised learning can be broadly divided into following main steps, i.e., training and testing. In the data preparation step, features are extracted (generally referred to as feature extraction) from the data set. The extracted features are further merged by using different strategies to produce unique features such as *n*-gram; which is essentially a word formed by the combination of more than one features. During the training stage, labeled data set, i.e., a set of inputs with their corresponding correct output classes (e.g., PUP and legitimate) is given to a learning algorithm. The learning algorithm tries to find the relationship between instances and their respective classes. The mapped relationships can be interpreted, and all the relationships found in a single data set are summarized in the form of a model called classifier. The generated classifier is tested for predicting the class of novel instances. The performance of classifiers is commonly affected by the data preparation step. This work contributes from data preparation stage to testing stage. Different data preparation and data reduction (for the purpose of using most relevant data) techniques from the field of text mining are investigated, and new techniques are also proposed. For the testing stage, a model is proposed, which uses veto voting to incorporate the decisions of different learning algorithms.

The thesis consists of two logical parts. The first part contains the background, primary research contributions, research approach, and conclusions. The second part consists of four published papers. The first article focuses on spyware; the second paper focuses on scareware detection. In the third publication, the work on adware detection is presented while the fourth paper presents a generic framework for malware (especially PUPs) detection.

Two

Background

2.1 Malicious Software

John von Neumann presents the concept of self-replicating programs during a presentation in 1949. It is eventually published in a book [14]. This concept is considered as the theoretical foundation for malicious software (malware). The first example of self-replicating programs is the "*Creeper*" that appeared in 1970 for ARPANET [15]. However, creeper is considered as an experimental benign program. For personal computers, Elk Cloner (released in 1981 for Apple computers) and Brain (released in 1986 for IBM computers) are considered first malign programs. Since then malware are in the wild, i.e., outside the computer system, or lab and software are divided into legitimate and illegitimate (malware) categories. Traditionally, malware are developed for entertainment, to earn fame (through worm), or due to a political issue [16]. Later, their power is realized to create the malicious behavior for the destruction of data. Up to the end of nineties, malware family includes only traditional malware that stealth their presence [17]. With the availability of the Internet in the mid-1990s, for the general public, new generation of malware, i.e., potentially unwanted programs appear in the wild with different objectives such as to invade the privacy of users and earn the revenue. The PUPs are also referred to as privacy-invasive software [18] or potentially unwanted applications/software/technologies [19]. This thesis will use the term PUP in subsequent sections.

In 2000, presence of the first type of PUP, i.e., spyware is announced [3]. Spyware are used to obtain behavioral and personal information about users. The collected information may be used for different purposes such as to create the user profile for market segmentation or to cause financial losses such as credit card fraud [20]. Depending on their functionality; spyware can be broadly categorized as useful spyware, and malicious spyware. Most of the spyware are PUP from the users' point of view; however, some software vendor may not be in accord with users' point of view. An example of such software is " DoubleClick " [1], which is considered useful for commercial organizations to track the users' advertisement related surfing habits. If a spyware steals the user's personal information such as financial information or creates security risks such as add a covert channel for the communication with the system, it is considered as malicious spyware [21]. Another type of PUP, i.e., adware, also emerged at the same time with spyware. Adware invade the privacy of a user by monitoring his or her surfing activities, or preferences in order to display customized pop-up, or pop-under advertisements for that particular user [22]. New generations of adware are also capable of reading data from locally stored files, collect surfing or chat related information, and make remote connections for transferring and installing software in the future [23]. Spyware and adware are used for indirect revenue generation as both collect the information and then this information is sold to a third party. In 2003, a new type of PUP, i.e., scareware appears that is used for the direct revenue generation[2]. Scareware represents scam applications that usually masquerade as security application such as anti-malware software, more specifically anti-virus software. Scareware scare the users about the security of their system and tricks them to pay money for their security services [24].

The status of potentially unwanted program, as legitimate or illegitimate, depends upon the user's perspective. Some software such as

[1]http://www.google.com/doubleclick/

[2]http://www.secureworks.com/research/threats/popup-spam/

remote administration tools, packet sniffers, and key-loggers may be considered illegal from the end user's perspective. However, a system administrator may require them on end users' system as the part of company policy, and that makes their presence legal. On the other hand, some users may ignore the existence of a PUP with the legitimate software since it may provide some useful functionality either on its own or by the accompanying software such as an adware software can be used by the user for getting commercial messages. However, if a PUP is stealing the financial information of the user such as credit card number, bank credentials, and relaying it to the third party without the user's knowledge, it will be considered illegal. To shift the status of PUPs from illegal to legal boundaries, the presence of a PUP and actions performed by that particular PUP may be mentioned in the End User License Agreement (EULA) using the verbose legal language that a novice user cannot interpret [25]. Due to their presence in EULA and to avoid legal consequences, anti-malware vendors are reluctant to indicate the presence of PUPs. There are many commercial inducements, such as income for advertisers from online ads distribution through adware, which has elevated the PUP industry [18]. Therefore, it is reasonable to assume that the volume of this industry will increase even more in the future.

2.2 Malware Detection

The malware detection problem is addressed by using either the proactive approach or the reactive approach. In the proactive approach, vulnerabilities from the system are eliminated, and in the reactive approach an action is taken after the malware appears in the system. The reactive approach, in which malware detection is the primary method of protection, is commonly used by anti-malware software vendors [26]. Reactive approach is further extended by the dynamic detection or static detection. In the dynamic detection, an emulation in the system is used to identify the malware [27]. Advance and sophisticated malware are capable of detecting the emulation and can elude it [28]. In the static detection, signature-based approach or heuristic-based approach is used for malware detection. However, the backbone of both approaches is a basic pattern matching mechanism, i.e., comparing the features of files to a database of known malware features [26]. The signature-based approach revolves around the use of signature database that contains byte strings which are unique to different instances of malware [29]. The heuristic-based approach, on the other hand, uses a behavior threshold for the malware detection. For both methods, anti-malware vendors need to get novel instances, analyze them, create new signatures or rules, and then update their databases [26]. On the other hand, after releasing a malware, the malware author waits befor unleashing the variant of that particular malware, until the signature or rule database is updated by anti-malware vendors for that particular instance [30]. It is reported that malware authors are recycling the older malware[3]. Moreover, when software vulnerability is announced, the interval between the announcement of a vulnerability and unleashing the malware who exploits that particular vulnerability is decreasing [31]. An average of 10 hours time is required by the anti-malware vendors to respond to the latest released malware, variants or recycled malware [12, 32] and create the signature or the rule for that particular malware instance. Consequently, anti-malware signature/rule database is growing regularly [12]. Due to this reason, these approaches are considered inadequate against the zero-day attacks [33, 34].

Most detection approaches are developed in order to detect the traditional malware especially viruses and worms. PUPs are not considered harmful; therefore, the PUP detection is not investigated thoroughly. The problem of PUPs is currently addressed through the static detection, i.e., signature-based approach and heuristic-based approach. However, it is not sure whether these detection methods are capable of detecting PUPs because PUPs do not employ any particular routines like traditional malware or do not contain malicious payloads [10]. On the other hand, PUP authors apply a number of stealth techniques to avoid de-

[3]http://www.microsoft.com/en-us/download/details.aspx?id=5454

tection. Due to variation in the behavior of different PUP groups, alternative ways of being undetected are applied. To avoid signature-based detection, they alter their code, manipulate indicative features that may be used as the signature, create new variants and even new families. For counteracting the heuristic-based approach, PUP authors blend the legitimate behavior with the behavior of their application or mimic the legitimate behavior as much as possible [6].

Due to the use of various stealth techniques from malware vendors especially PUP vendors such as code obfuscation, the trend of malware detection is shifted from detecting malicious code in a file to classify a file as malicious before the file is executed. For a strong and effective defense, it is essential to enhance the efficiency of malware detectors and to generalize detector's ability to detect the known malware as well as the zero-day attacks. This situation warrant the application of knowledge from other areas for the malware detection. Recently, the use of artificial intelligence, statistical methods and machine learning (ML) has been investigated to extend the defense mechanism. Previously, machine learning is applied mainly for text mining and has shown promising results. The ML methods such as supervised learning are applied to automate and extend the heuristic-based malware detection. For the supervised learning, a computer file is considered as a stream of text. The stream of text can be represented by different representations such as byte codes or assembly instructions. This stream contains different patterns that can be extracted using different strategies. These patterns are further used as *n*-gram to input the knowledge about the malicious behavior in the learning methods. The input knowledge is usually given as a batch of different examples. The batch is further used to build a model or the learned system. The learned system is further applied on the unknown examples for their classification.

2.3 Terminology

Following are the terminologies that are related to the thesis. These terminologies can be defined in several ways to address different requirements according to different parameters. The following definitions of these terminologies are relevant to the primary focus of this particular thesis, i.e., the PUP detection that endangers the privacy of novice users.

Privacy as a concept is initially discussed by the Greek philosopher Aristotle who emphasized humans need to maintain social/ public life (i.e., to be part of society) and private/ personal life [35]. The private life of a person is considered as the basic human right and is called the privacy. The first literary definition of the privacy is presented in an article " The Right to Privacy " [36]. According to the first definition, privacy is:

"*The right to be let alone*".

With the technological development, a novice user is more concerned about the privacy of the data or the information present on the computer; thus the concept of privacy has been further extended as following [7]:

"To control, edit, manage, and delete information about them [selves] and decide when, how, and to what extent information is communicated to others".

The word privacy in this thesis refers to the information privacy, which indicates the legal right of a person to decide about the collection, storage, and sharing of personal information in the digital form. The user can surrender the legal right of the privacy partially or fully by providing an informed consent to the other user or a software.

Informed consent is a legal term that is derived from two words, i.e., informed and consent [37]. The term informed means that the accurate information is provided to the user who is able to accurately interpret that specific information. The term consent refers to an opportunity given to a rational user to accept or reject something offered to him without any pressure or lure. It is a common practice for the software vendors to obtain the informed consent of users by providing the information in the EULA. End User License Agreement is a software license agreement between the software manufacturer and the user who describes the permissions, rights, and restrictions about the use of that particular software. In the case of PUPs, the information about the presence of an additional software or full functionality of the software is concealed in the EULA. The obscure information in EULA sets new challenges for the traditional malware detection methods. These new challenges can be addressed by applying data mining and machine learning.

Data Mining (DM) is the process of analyzing electronically stored data by automatically searching for interesting patterns [38]. Learning algorithms use the discovered patterns for the analysis and the prediction. DM and ML overlaps in many areas such as DM uses ML algorithms for predictions, and ML algorithms use DM methods, e.g., preprocessing step to improve the predictive accuracy [38]. However, the difference between two fields is clear as DM focuses on the discovery of new patterns in the data; ML learns from patterns in the input data to provide the prediction.

Machine Learning is the branch of artificial intelligence that is concerned with the design and development of algorithms that allow computer programs to learn and evolve their behavior or state by discovering patterns or relations in data [38]. The discovered patterns or relation in the data is further used to generate input data set, and learning algorithms can be applied to the input data set to develop a model (i.e., a classifier) or a regression function to solve complex problems. This is formally described as [39]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

In the context of PUP detection, the definition of [39] can be elaborated as following:
Task T: Recognizing and classifying PUPs from the set of legitimate and PUP files.
Performance Measure P: Percentage of PUPs and legitimate files correctly classified.
Training Experience E: A data set of PUPs and legitimate files features with their known classes.

ML can be divided into either supervised learning or unsupervised learning. In supervised learning, labeled data set, i.e., a set of inputs with their corresponding correct output classes is given to learning algorithms [38]. The correct output is used to evaluate the predictive performance of the learning algorithm. If the data is incompletely labeled, unsupervised learning is used to cluster the data set into groups having similar features.

Data Set in ML refers to the collection of the data in the tabular form to represent a schema and set of instances that corresponds to schema [38]. Generally, columns in the table correspond to variables; rows represent members or instances (a single object for either learning a model or applying a model). Most of the ML data sets are single-fixed format tables, which describe instances by their feature vectors where a feature is the specification of an attribute and its corresponding value. Features from a particular instance in the data set are extracted either as a single feature set or by combining several features. This process is commonly referred to as feature extraction. For the PUP detection, feature vectors are used to represent the PUPs or legitimate files.

File Representation: For malware classification, data sets are prepared by using various representations of binary files. A binary file can be represented by specific type of extracted features that are either present in the file or features obtained from any kind of meta-analysis (for example, runtime generated digital footprints). The extracted features may be further used to create *n*-grams, which in turn can be viewed as words or terms if the learning problem is defined as a text categorization problem.

N-gram: In the domain of text categorization, *n*-gram is a contiguous sequence of *n* items, *n* characters, or *n* words. The *n*-gram is a fixed size string, but may have variable length if the basic units are not having the same length such as if words are used to create *n*-gram, each *n*-gram will have fixed size, but may have variable length. The size of *n*-gram may be greater than or equal to one. The feature type used to create *n*-gram affects the performance of the generated model or the classifier.

Classifier: A learning algorithm tries to find the relationship between instances and their respective classes by using different strategies, such as decision tree, depending on the family of algorithm [38,39]. The mapped relationships can be interpreted, and all the relationships found in a single data set are summarized in the form of a model called classifier. Later, the classifier can be applied to other data sets, where class value of instances is unknown, to obtain a prediction about the target class for each instance in the data. This process is referred to as classification.

2.4 Related Work

In the end of 80s, theoretical evidence is presented to demonstrate that accurate malware detection is impossible [40, 41]. However, some security experts argue that this may not be an actual case in the real-life [42]. Malware detection is gone through different phases, which are from signature-based detection to behavioral-based detection [43]. Initial malware, i.e., virus are used to infect files, boot sectors, and are easy to detect using signature-based detection methods up to the mid of 80's. With the development of more sophisticated malware such as the worm in the mid of 80s, [44] and Trojan horses, which exploit the vulnerabilities in the operating system and/or software such as web servers, more sophisticated detection techniques are required. Heuristic-based detection and Behavior-based anomalies detection techniques are used to address the challenge [45,46]. From the 2000-2005, the highest number of reported malware are PUPs [47]. Till that time; malware industry generally relied on signature-based detection technique [48]. Signature extraction is not a new or novel approach to address a similar problem. This approach has shown promising results in the less controlled environment such as honey pots and other threat collection systems [49,50].

Some other techniques such as generic exploit blocking for unseen instances are investigated [51,52]. In another effort, creation of the generic signature for a family of malware is also investigated [53]. However, these techniques face the challenge of execution speed, CPU load and false positive rate. Moreover, for the success of these techniques, vulnerability knowledge is required in advance, which may not be available. To address these issues, especially prior vulnerability knowledge, test cases are generated, which produce models [54,55]. The generated models are further used to identify vulnerabilities. However, these models are also used by malware vendors to fabricate evading techniques [47].

To generalize the known malware detection and detect unknown malware especially zero-day attack, use of data mining and machine learning methods is first investigated in 1994 [56]. The use of supervised learning techniques in ML for malware detection is first investigated in 2001 [33]. In the training phase of supervised learning, a batch of malicious and legitimate examples is provided to the learning method. The batch of examples contains the patterns that are used for representing executable files (i.e., file representation) [57]. These patterns are generally referred to as file features, which are extracted from the executable files by applying different reverse engineering techniques. The file features are considered static features (morphological) and supervised learning is usually applied on extracted file features to extend the static analysis [57]. The researchers in the malware detection are primarily focused on the static feature extraction (commonly referred to as malware signatures) from executable files and their analysis, and not on dynamic (behavioral) features. Static features are typically used to generate the *n*-gram. The concept of *n*-gram is widely investigated in the domain of text categorization [58]. The *n*-gram probability distribution and *n*-gram models have shown promising results, independent of any specific language, both in natural language processing and text mining [59]. The *n*-grams are also used for authors profiling, text comparison, and speech recognition [60]. The concept of *n*-gram is also applied for the malware detection in the form of *tri*-grams extracted from the virus signatures [56]. *N*-grams are also fed into the neural network based classifier for detecting viruses [61, 62]. Similarly, several neural network based classifiers are also generated on the *n*-gram for the virus detection [63].

Static features of executable files, i.e., byte code, DLL information and strings, in the form of *n*-gram, using supervised learning are also investigated for malware detection [33]. Byte *n*-gram is a fixed size sequence of byte code in the hexadecimal format, extracted from the binary files. This approach is further extended evaluating different sizes of *n*-gram representation under different settings [64]. Different sizes of byte code *n*-grams are also used to create profiles of legitimate and malicious classes [65]. New malicious instances are assigned to the most similar profiles. The work of [33] is also extended by applying different feature selection, and learning algorithms [66]. The results produced are better than the work of [33]. In another study, class imbalance problem is also investigated for the malware detection [67].

Byte code *n*-gram representation is extensively investigated for the malware detection. However, main focus of investigations is to classify viruses, Trojan horse, worms and legitimate software [33, 65, 66]. However, byte code features (being a small piece of hexadecimal code) are not capable of providing any interpretable and useful information, which can be used for understanding the behaviour or the function of executable files [68]. Moreover, it is also difficult to trace them back in the original software and understand the purpose of a particular byte code or sequence of byte codes [68]. However, this representation is widely investigated for malware detection due to its high success rate. To address the deficiencies in byte *n*-grams, the use of operational code (opcode) n-gram, and sequence of opcodes is also investigated for representing executable files [69]. An executable file is disassembled to obtain assembly instructions and opcode is the part of assembly instructions. Opcode representation is used in two different ways to make n-grams, i.e., by having fixed length sequences [29, 70] and by having variable length sequences [24, 71]. Variable legth opcode n-grams are normally produced to capture an entire function in the executable file.

Initially opcode are used to generate signatures for the detection of different variants of worms and some types of spyware [69]. To avoid the problem of manually updating the signature databases of the scanners; data mining algorithms are used as part of a scientific study to create a generic scanner [16]. In this study, experiments are performed on two different data sets: the first data set contains the opcode of each instruction, and the second data set contains the opcode as well as the first operand of each instruction. The frequency of occurrence in the virus class, and in the legitimate class is used as a basis for feature selection. The frequency of opcode distribution is also used to isolate malware from legitimate. Opcode based n-grams under different settings along with the class imbalance problem are also investigated [29]. The idea of using variable length instruction sequences is conceived as a part of an attempt to detect worms. The valuable instruction sequences are used to create the data set, which is further fed into learning algorithms and ensemble learners to classify novel instances of worms.

Opcode sequences are investigated by different researchers to understand the behavior of malicious programs (mostly viruses) and derive the distinct patterns of behavior, which separate malware from legitimate programs. Most of the researchers have used the commercial disassemblers such as IDA[4], and PE Explorer[5] to disassemble binary files. The commercial disassemblers are capable of understanding the file structure. Researchers have used the opcode from the specific sec-

[4]http://www.hex-rays.com/products/ida/index.shtml
[5]http://www.heaventools.com/

tion of the file, i.e., code section. In comparison of two kinds of opcode *n*-grams, variable length sequences produce better results in terms of detection rate and capturing a particular function. The main advantage of using opcode sequence is that opcode sequence can provide the meaningful information and can be traced back to its location in the original file for further analysis.

Except the *n*-gram representations, the executable files also contain other meaningful static features in the form of printable strings, meta data about functions, and meta data about the file. These strings are stored in different parts of files. The examples of these features are messages shown by files and functions calls. These features may give a clue about the activities performed by the executable file and are used for malware detection [33,72,73,74]. Albeit, these representations achieved satisfactory results, such representations may not be considered reliable for malware detection problem because it is easy to change this information by changing the compiler or packager or class files.

Three

Primary Research Contributions

A survey of the research, in the domain of malware detection using ML, in the last decade is presented in [57]. The research summary provides an overview of different types of features for representing executable files. Different types of representations along with possible features extracted for the particular representation is shown in Table 3.1. The research in the last decade is heavily influenced by the use of byte code *n*-grams or byte *n*-grams, followed by the fixed length opcode *n*-grams. However, following problems can be ascertained from the last decade research:

1. Most of the researchers have used the same data set of viruses, worms, and Trojans horses obtained from the website VX Heavens[1].

2. Similar procedure for *n*-gram extraction is used, and only few researchers have addressed *n*-gram extraction techniques.

3. Feature selection methods have not been investigated. Mostly, traditional feature selection methods from text mining are borrowed by the researchers.

[1]http://www.vx.netlux.org

Table 3.1: Different static features used for the malware detection.

Type	Feature
Byte *n*-grams	*N*-grams consist of a sequence of bytes in the hexadecimal representation.
Functions	Meta data about functions in executable files.
Opcode *n*-grams	*N*-grams consist of a sequence of the assembly language operations (opcode), e.g., ADD, MOV, etc.
PE-Features	Meta data about executable files, e.g., creation time, structure, etc.
Strings	Readable strings from executable files.

4. For obtaining the meaningful static features such as opcode, researchers have used commercial software, and features from a specific part of the files are investigated.

3.1 Aim and Scope

The thesis aims to enhance the heuristic-based detection methods for potentially unwanted programs by using supervised learning. It is argued that computer programs, i.e., executable files contain patterns in terms of functionality. The patterns in the executable files can be converted into the *n*-grams, which can be further used to indicate the purpose of the program and classify the program. By providing information about the purpose and class of a program, users are empowered in their consent about using that particular software. The scope of this thesis is limited to potentially unwanted programs, which fall within the grey zone of the legitimate programs and illegitimate programs.

3.2 Contributions

The following sections elaborate the contributions of this thesis in general and particular contributions of the published articles that constitute

Chapter 6 to Chapter 9. The details of contributions in each chapter according to the research questions (R.Qs) (given in section 4.1) are as follows:

Chapter 6 presents a spyware detection approach that investigates the possibility of using DM and ML based detector to detect spyware. The data set of binary features (i.e., fixed size *n*-grams) representing the real-world situation of malware and legitimate software is prepared with different configurations. Two feature reduction methods are proposed to obtain a subset of data. Five different supervised learning algorithms from different families such decision tree based are applied on all the configurations to generate classifiers. The proposed results suggest that proposed detector is capable of detecting the spyware in both cases, i.e., either it is embedded with other executable or as a stand-alone application. The detector is also capable of indicating the presence of spyware, when the malicious files are scarce. The results also suggest that larger *n*-gram size is a better option for the PUP detection. This chapter contributes to R.Q. 1.1 in that the results suggest the use of byte code features for the spyware detection. However, it is worth to note that byte code features are unable to provide the useful information about the behavior of the program. The other contribution of this chapter is to R.Q. 2.1 in that two proposed feature reduction methods are investigated, and the results suggest that the features reduction using frequency of a feature improves the detection results.

Chapter 7 proposes an Adware detection algorithm. To investigate the validity of the proposed algorithm, extracted sequences of opcodes are used as the primary feature for classification. Feature selection is performed using different configurations, to obtain 63 data sets. In order to obtain an efficient and accurate detector, six data mining algorithms are evaluated on these data sets. Empirical results show that the proposed method can be used to accurately detect both novel and known adware instances. This chapter contributes to R.Q. 1.2 as the experimental results indicate that opcode sequences are the best choice for the PUP detection than the byte code *n*-grams. The results also extended the results of previous study that larger *n*-gram size is a best choice for the PUP detection. The Chapter 7 also contributes to the R.Q. 2.2. For the R.Q. 2.2, the effect of feature selection is evaluated in two stages, i.e., primary feature selection using the frequency of a feature and secondary feature selection using the statistical importance of a feature, to find an optimal percentage of features.

Chapter 8 extends the traditional heuristic detection methods to an automated method for extracting the behavior of a particular type of PUPs (i.e., scareware), and legitimate software in the form of variable length instruction sequences. Extracted fragments of behavior are analyzed in order to improve the existing knowledge about the scareware detection. This chapter contributes to R.Q. 1.2 in terms of the features used. The results suggest that the use of variable length *n*-gram is better than the fixed size *n*-gram as they also assist in capturing a complete function in the program without breaking the function after a specified length. However, results from both Chapter 7 and Chapter 8 are in favor of using *four*-gram. The *four*-gram size is capable of capturing the complete valuable functions, which can be further used as *n*-gram for the PUP detection. This chapter also provides pointers for the R.Q. 2.2 as the primary feature selection and secondary feature selection from the text mining are evaluated. Another major contribution of this chapter is to the R.Q. 3 as the experimental results proposed that combining the inductive biases of multiple algorithms improves the detection rate.

Chapter 9 presents a generic malware detection approach that is tested for a specific type of PUP, i.e., scareware. This chapter proposes two data selection techniques (feature extraction), an ensemble learning algorithm, and an empirical evaluation of the proposed algorithm. The results of this chapter contributes to R.Q. 3 by providing the empirical evaluation of the impact of combining inductive biases and the effect of different voting schemes on the output of the ensemble. For the R.Q. 3, results also suggest that different voting strategies shall be adopted for different tasks according to the problem in question. The results also contribute through the pointer that combining the inductive biases of several algorithms trained on different representations predicts better for the malware detection than combining the inductive biases of different algorithms trained on the same data set.

The overall contribution of the thesis is to provide an understanding about the automation of PUPs classification. Potentially unwanted programs resemble legitimate software in terms of functionality. Thus, it is difficult for human experts to classify a particular instance as PUP. This thesis introduces the idea of using a non-commercial disassembler for extracting either the n-gram or the behavior of a particular software, which can be further used for identifying the class of software. The experimental results indicate that instruction sequence is a better option in comparison of other representations such as hexadecimal n-grams. The experimental results also indicate that either a technique is adopted for extracting instruction sequence based n-grams or behavioral n-grams, the n-gram size four is a suitable option for the PUP classification. The experimental results indicate that relative frequency of a feature is better option in comparison to the frequency of a feature. It is also suggested that a suitable voting scheme will enhance the accuracy of the PUP classification system.

Four

Research Approach

In this thesis, the PUP detection problem is addressed through different techniques of data mining and machine learning. On the abstract level, the process of applying DM and ML can be divided into the following steps [75]:

1. Problem Statement
2. Data Preparation (Data Selection)
3. Data pre-processing or Transformation
4. Model Generation (i.e., model estimation (training phase))
5. Validation (i.e., model interpretation; analyzing the quality of the model)

For the first step, the problem statement is clearly stated as the PUP detection problem. The main contributions of the work are in the remaining four steps, and these contributions are experimentally proven. The thesis contributions (section 3.2) also provide the answers of the research questions in the section 4.1. For the step 2, the main contribution is to investigate the different features of a binary file representation for the PUP detection. This thesis also introduces the idea of using overlapping n-grams and adjacent-nonadjacent n-grams extracted from the binary files as the primary feature for the file representation. After extracting all the features from a data set in the required representation, all the features may not be significant for the detection. The data pre-processing step, i.e., step 3, is required to select the relevant features. The step 3 in the process is the most computationally intensive procedure and performance of the step 4 and the step 5 may also depend upon it. Therefore, for the step 3, uses of different feature reduction methods from text mining are investigated. For the step 4 and step 5, an algorithm is proposed, which combines multiple learning algorithms and uses the veto voting to predict the class of an instance. This thesis compares different feature extraction techniques, feature sizes, feature selection settings, and performance of different algorithms upon different settings to find an optimal solution.

4.1 Research Questions

Potentially unwanted programs may mislead a novice user about their actual functionalities by hiding their existence as the part of a complete software package or by indicating their intended functionalities in the End User License Agreement. Users usually ignore the long written terms and conditions of the EULA and accept the EULA without reading. This behavior makes the user consent questionable, as a result the malicious software authors may claim the legitimate status for their application. Due to this reason, traditional countermeasures may ignore the presence of known PUP instances. Traditional measures are also incapable of detecting zero-day instances. Consequently, new detection methods that are capable of detecting known instances and the zero-day attacks are required. The usage of ML and DM technologies is investigated for traditional malware detection. Supervised learning in ML has shown promising results in the detection of viruses, worms, and Trojan horses. However, supervised learning is not investigated for the detection of PUPs.

Every binary file contains different features that can be used for the file representation in the data set. For example; the name of a file is a feature, which represents a file in the set of files. Other features are file size, file creation/modification date, instructions in the file, messages in the form of printable strings in the file and other important information available as part of the file, e.g., information in the file header. However, all the features of a file may not be suitable for the file representation due to different reasons such as the file name does not provide any useful information about the function of file. In different studies, for the detection of viruses, worms and Trojan horses, different file features obtained directly from the file or from the file conversion are used. The common file features used in the experiments are printable strings, meta-data about the functions, byte code (in hexadecimal form), and extracted assembly instructions. However, it is obscure that similar file features may also be used for the PUP detection as PUPs have subtle difference with the legitimate programs. Hence, it is required to find the file features, which can help to achieve optimal results in the PUP detection and provide relevant information for the analysis of PUPs behavior. Different potential file features or combination of file features can be investigated to find the appropriate feature representation/s for the PUP detection. Hence; this thesis investigates, *which file features are suitable for detecting potentially unwanted behavior in an executable file?* This investigation addresses a particular dimension of the problem in hand and provides the answer to contribute in the overarching aim of the thesis. Thus, this particular dimension is further addressed by following two sub-questions.

R.Q. 1.1. *Can hexadecimal based features of a binary file be used for the accurate PUP detection*?
This research question is investigated in Chapter 6, which describes the experimental results for the PUP detection based on byte code features in the hexadecimal form. Chapter 6 also investigates the suitability of combining multiple features for producing a unique feature, i.e., *n*-gram. The byte code features may also be extracted in the binary or octal representation. However, the detection result will not be different from the hexadecimal representation. This chapter provides the answer about the suitability of byte code (hexadecimal form or similar forms) based

feature; however, this chapter does not address the suitability of other representations such as instruction sequences.

R.Q. 1.2. *Can assembly instructions be used to descry PUP behavior and as the file feature for the accurate PUP detection*?
The answer of R.Q.1.1 indicates that the combination of features to produce a unique feature is a better option than using a single feature. The suitability of instruction sequences or the specific part of the instructions in sequence for the PUP detection and strategies for their combination are investigated in the Chapter 7, Chapter 8, and Chapter 9. Different feature combining strategies are investigated to produce unique *n*-grams, which provides valuable information about the function or the behavior of PUPs. The identified function or behavior can be further used to accurately detect PUPs.

The features obtained from a binary file may vary in numbers such as the file name is the only one feature of a file. However, if a file is disassembled, the number of generated features will depend on the file size. If a data set is prepared with all the generated features, that data set cannot be processed directly by most of the machine learning algorithms. Consequently, a method is needed to reduce the number of features significantly in such a way that it does not degrade the predictive performance of machine learning algorithms. During the feature reduction process, valuable features from the original feature set are selected to generate a feature subset. The generated subset shall be able to provide an understanding of the data set. In text mining different methods such as frequency of a feature in a particular document or the whole data set, or statistical importance of a feature in the document, are used to find the valuable features in a feature set. The PUP detection task can also be viewed as a binary text classification and correlation of each feature with its class can be computed by using the frequency of the feature. Further, the frequency of each feature may be used to find the subset of valuable features. However, it is worth to investigate that *What is the impact of frequency based feature selection methods on classifier performance for the PUP classification?* To find the solution for this question, this question can be split into two further questions:

R.Q. 2.1. *Is the frequency of a feature a suitable measure to select the valuable feature*?
To find the answer of this R.Q., frequency based feature selection methods are investigated in Chapter 6. The proposed frequency based feature selection methods investigate the suitability of common features and frequency of features in the files. However, a feature may also have a relative frequency, which is not addressed throughout this study.

R.Q. 2.2. *Is the statistical importance (weight) of a feature a suitable measure than the frequency of a feature to select the valuable features*?
To find the solution of this question, different feature selection methods, which calculate the composite weight of a feature, are investigated in Chapter 7 and Chapter 8. The Chapter 7 and Chapter 8 also investigates the possibility of extending the feature selection in multiple, i.e., primary feature selection and secondary feature selection steps where each step applies a unique method to calculate the relative weight of a feature to produce the subset of valuable features.

The subset of valuable features and feature representation impregnate the inductive bias of a learning algorithm and affect the performance. Due to different feature representations, generated models may perform contrary. However, different models can be combined to increase the overall predictive performance. The combination of different models to reach final predication is usually referred to as an ensemble. Ensemble can use different methods such as voting, to reach the final prediction. This element is investigated by the following given research question.

R.Q. 3. *What is the impact of a voting strategy on the outcome of an ensemble for the PUP detection*?
An appropriate combination of inductive biases may increase predictive performance as well as an inappropriate combination may cause a significant decrease. Diversity among the models in the ensemble enhances the predictive performance of the ensemble. The strategy to select the

appropriate finite set of models and method to achieve a collective decision can be either one of the methods already in practice or a customized method. An example of methods in practice is the bagging. In bagging each algorithm provides its prediction about the instance (all the predictions are weighted equally) and final prediction of the ensemble is given on the basis of majority of votes. This thesis investigates the use of an ensemble for the PUP detection in two steps. In the first step, Chapter 6 and Chapter 8 investigate an ensemble in practice; and compare the ensemble's performance with the performance of other learning algorithms. The customized method of combining the inductive biases such as veto voting can be adapted according to the problem in hand. Therefore, in the second step, Chapter 9 investigates two customized voting strategies, i.e., veto voting and the trust-based veto voting and compares them with the majority voting.

4.2 Research Methodology

The selection of the research methodology is a crucial step because it helps to learn the cause and the effect relationship for the problem in hand. This thesis performs the empirical investigation of the problem in hand, which is referred to as quantitative research. The quantitative research can be carried out by using different research designs such as the experiment, and the case study. This thesis applies experimental design to answer the research questions in the section 4.1.

Experiment method is also referred to as the scientific method [76]. Experiment is defined as follows [77]:

" *A study in which an intervention is deliberately introduced to observe its effects* " .

An experiment is performed to determine the *cause-effect* relationship between the variables. In the experiment all variables, except one variable, i.e., independent variable are controlled. Due to controlled variables, experiments are also referred to as controlled experiments [78]. The other variables involve in a controlled experiment are dependent variable, control group, and constants. All the variables of an experiment are explained according to the thesis following:

- *Independent Variable* is the variable, which this thesis measures, generally referred to as cause or experimental conditions or treatments. The independent variable in the thesis is the number of valuable features.
- *Dependent Variable* is a variable, which is dependent upon the independent variable, generally referred to as the effect. The dependent variable in this thesis is the percentage of correct decisions by different algorithms.
- *Control Group* is a group, which receive either no treatment or a standard treatment and is contrasted with the experimental group. This thesis uses ZeroR algorithm as a control group and the performance of all the algorithms is compared with the ZeroR algorithm.
- *Constants* are variables, which are not changed during the experiments; they may not have any direct effect on the outcome of the experiment. The constants for the thesis are operating system, Weka software, feature extraction and feature selection routines and learning algorithms.

For any particular experiment in the thesis, the independent variable is dependent upon other parameters, i.e., number of files and the file size. This independent variable has positive correlation with the dependent variable, i.e., percentage of correct decision, which implies with the increase in the importance (quality/value) of features, the percentage of correct decisions also increases. It is common in the field of machine learning to have a larger data set (that lowers the variance across the different data sets) to increase the accuracy. The likelihood of a true causal relationship between the independent variable and dependent variable for the experiments is high. The increase in the independent variable, i.e., increase in the number of valuable features, up to a certain extent, may help in indicating the class of files, (considering the effect of constant, i.e., feature selection routines) and the learning algorithms will be able to predict more correctly.

By using the variables mentioned above, two types of experiments, i.e., *Random Experiments* (RE) and *Quasi Experiments* may be performed. In RE, random assignment of experimental units to the treatment is made, no prediction about the outcome can be given. However, the outcome of experiments always lies in the list of possible outcomes. A random assignment of the experimental units helps in deriving a stronger conclusion about the *cause-effect* relationship between the variables. Quasi Experiments are similar to the random experiments except that the quasi experiments lack random assignment of experimental units to the treatment [77]. In quasi experiments "*cause is manipulable and occurs before the effect is measured*" [77]. For the PUP detection experiments, randomization may not be required as ML algorithms are independent of data collection methods. However, the predictive performance of an algorithm needs to be compared by using controlled experiments. Chapter 6, Chapter 7, Chapter 8, and Chapter 9 compares the predictive performance of ML algorithms. Chapter 7 and Chapter 8 also investigate the *cause-effect* relationship. Chapter 7, Chapter 8, and Chapter 9 also investigates the impact of constant, i.e., feature selection routine.

4.3 Research Validity

Validity concerns the correctness level of the outcome of an experiment and its applicability in the real-world settings or righteousness of the answer of a scientific problem [79]. For an experiment, there are always possible threats to its validity that can be divided into four principal categories. Validity threats include internal, external, construction and conclusion; where each type may address a specific methodological question [79]. Each category may be further divided into different sub-categories; if needed.

Internal Validity threats are related to experimental settings. Most internal validity threats concern the effect of an independent variable on the dependent variable in a controlled experiment and the effect of any change in the experimental setting to its outcome [80]. Different variables such as history, maturation, testing, instrumentation, statistical regression, selection, and experimental mortality interfere with the internal validity [81]. For the experiments in the thesis; most of these variables are not applicable. The related internal validity threats are concerned with the selection, i.e., random assignment of application during the training phase and the percentage of features used. To mitigate the selection threat, statistical methods are used, which are addressed in Chapter 6, Chapter 7, Chapter 8, and Chapter 9. The other internal threats are related to the experimental procedures, i.e., extraction of features from the file. For the feature extraction, extraction routines may not be able to extract features from all the programs. The threats for the experimental procedure are mitigated by using different representation and addressed in all subsequent chapters.

External Validity threats are concerned to the generalization of conclusions or results from a particular experiment to the more general population. The general research question addressed by the external validity threat is the criteria to apply results of the experiment (as experimentation is done in a controlled setting by creating the artificial environment) or the conclusion drawn from the study of another similar population or in the real-world settings [80]. For the malware detection experiments, malicious file percentage in the wild is an example of the external validity threat that is investigated in the Chapter 6 and Chapter 7. Another external validity threat for the thesis is that the data set used in experiments may not have representation of all families of malware in the sample, so the results may not be generalized. Correct sampling allows generalization and mitigates the external validity threats; statistical sampling and replacing methods are used in experiments.

Construct Validity threats are threats to the experiment itself and threats to the data preparation stage in the experiment; hence they assess the quality of experiment design [79]. Construct validity threats are mitigated by performing the statistical analysis of the experimental procedure and the relation between the variables [81]. For evaluating the relation between variables; variables shall be defined in a quantifiable way. These threats are not applicable to the experiments in the thesis. The followed experimental procedures and data preparation methods are standards within the domain.

Conclusion Validity threats concern the reasonability of conclusions drawn from the experiment [79]. In all the experiments in the thesis, statistical test such as the paired t-test is used. T-test is a parametric test, which is used to obtain the ratio of difference between means of two groups against variability between groups [80].

Five

Conclusions

5.1 Discussion

The PUPs are generally detected either by the signature matching or heuristic methods. The signatures and rules are kept in a database, which needs frequent and regular updates by the users. For obtaining the signature of a PUP or defining the rules to detect a particular PUP, a human expert needs to capture the PUP and manually analyze the PUP. ML and DM technologies are employed to automate the detection process from start to end. The features from the malware are extracted, given to learning algorithms for generating classifiers that are further used to predict the class of a file. The ML and DM technologies extend the heuristic-based detection method, and the generated models are suitable both for the novice user and the expert. For the users, the generated models may provide the information about the purpose of a program. Thus, users are empowered in their consent about using that particular software. This empowerment will enhance the privacy of users as users can decide to execute a specific file or not. However, it is impossible to give the final decision/recommendation about the execution of a particular file to the user as the definition of legitimate and illegitimate varies from region to region. These varying definition prospects the chances of malware vendors to lure the users by giving different options.

For the experts, this thesis presents an automated process to ease their job. The difference between a PUP and legitimate program is subtle, and human experts may not be able to track all the patterns in a file to conclude its behavior. The techniques presented in this thesis will help the experts to analyze the executable. However, one main drawback of the proposed techniques is that they are not capable of detecting a PUP, which contains encrypted malicious payload. The malicious part of the PUP cannot be decrypted without the key; thus, human intervention is required. At the same time, the presence of an encrypted module in a program can be considered as an indication of unwanted behavior. Another associated problem with the proposed technique is updating the generated classifiers. Although experimental results indicate that generated classifiers are capable of detecting the zero-day attacks; however, users need to update the classifiers due to the emergence of new examples and families of PUPs. However, this is different from updating the traditional signature or rules database where new signature or rules are added into an existing database on the daily basis. For the proposed techniques, to update the classifiers, classifiers are generated again on the new data set, which contains the new examples of PUPs. However, the classifier generation is a time-consuming task and shall be avoided at the user end. The task of classifier generation can be performed at a central location. Later, users can be informed about new classifiers and users can replace their classifier with the new classifier or this can be done by the system automatically. The size of this update is quite smaller than the size of new signatures added into the signature database, used in traditional detection techniques. Moreover, the frequency of new classifier generation and updating at the user end is also lower than the frequency of updating the signature database or the rule database.

This thesis investigates the classification (detection) of one particular family of PUPs in each experiment; however, experiments shall be extended for the multi-class classification. Moreover, the proposed technique can also be extended for the detection of PUPs in online social networks, as the basis of proposed technique is text mining. The thesis is not conclusive about a particular algorithm that can be used for the classification of all families of PUPs, thus several classifiers (from several algorithms) may be required for multi-class classification. An ensemble may be required to combine the decisions of multiple classifiers. Generally, the outcome of an ensemble is obtained by the voting of participants, thus, a voting strategy may also be required. This thesis investigates two different voting strategies; however, results may not be generalized for all the voting strategies. Moreover, most of the applied techniques are borrowed from the text mining, techniques devised specifically for the malware/PUP classification are required.

The proposed techniques extend the static detection that is used to classify a file as malicious or legitimate before the file execution. However, a malicious file may be misclassified and may be executed by the user. Moreover, users perform various tasks on the systems such as browsing. A user can browse through a website with the malicious contents such as malicious scripts. Such malicious contents cannot be detected by the static detection methods. Therefore, dynamic analysis and detection are also needed with the static detection. The proposed techniques are not tested on the malware using obfuscation. However, it is indicated that similar techniques are capable of detecting different variants of a malware in a family, which implies that proposed techniques are also capable of detecting the PUPs with obfuscated code.

5.2 Conclusion

Supervised learning can be performed by using a wide range of approaches such as artificial neural network, Bayesian statistics, decision tree learning, kernel estimators, support vector machines, ensemble of classifiers, and statistical relational learning. There is no single approach, which is suitable for all problems. More than one method can be used to address the problem. The decision of applying a particular approach is based on different factors such as availability of the data, the amount of the data and complexity in it, and dimensionality of the data. These all factors are also valid for the PUP detection. However, an implied benefit of the PUP detection is that PUP examples are available in the wild, which can be used to create the labeled data set for supervised learning. This leads us to address the critical components of the supervised learning process. Therefore, the aim of this thesis concerns about the applicability of approaches and has been achieved by addressing different dimensions, where each dimension is further addressed by a research question.

R.Q. 1.1 and *R.Q. 1.2* in the thesis address the feature representation dimension of the problem. Various features can be extracted from a binary file for the file representation. However, some features are easy to be customized by developers and may be considered unsuitable for supervised learning. Chapter 6 investigates the possibility of using byte code features for the file representation. Chapter 7 and Chapter 8 explore the possibilities of using opcode as a feature by using two different strategies, i.e., fixed size opcode sequence based *n*-gram, and variable length opcode sequence base *n*-gram. The results, presented in these chapters, suggest that for the PUP detection *four*-gram (*n*-gram size = 4) is an optimal size for both fixed size opcode *n*-grams, and variable length opcode *n*-grams. Chapter 9 proposes two different strategies to obtain byte code or opcode based features from the binary files, i.e., overlap *n*-gram, and nonadjacent *n*-gram. The results, presented in Chapter 7, Chapter 8, and Chapter 9 favor the use of variable length opcode sequences and overlap-*n*gram for the detection task. However, to understand the behavior of binary files, the results are in favor of variable length opcode sequences, and nonadjacent opcode *n*-grams.

R.Q. 2.1 and *R.Q. 2.2* concern the number of features used in the training of supervised learning algorithms. A huge number of features can be extracted from the binary file data set. However, all of these features may be unsuitable for the training purpose due to two factors, i.e., all the learning algorithms cannot process large number of features, and many features may not play a decisive role in the classification task. Therefore, it is essential to find the appropriate features by applying different strategies. Different text mining strategies are investigated, and two methods are proposed to address these R.Qs. Chapter 6 proposes two different strategies based on the frequency and common features for selecting the significant features. Chapter 7 and Chapter 8 explore the feature selection methods in two steps, and each step uses a different feature selection method from the text mining. The results suggest that a small number of features in comparison to the original feature set can be used to obtain the optimal performance for the PUP detection.

R.Q. 3 concerns the decision strategy for the class prediction of a binary file. As described earlier, supervised learning can be performed by using different approaches; however, the decision from each approach depends upon the underlying set of assumptions of the learning algorithm. Due to this reason, different approaches can predict differently for the problem in hand. To improve the prediction decision, ensembles are used in machine learning. Chapter 7 combines the inductive biases by using an ensemble algorithm random forest, which shows the promising result. However, similarly to other supervised learning algorithms, similar ensemble cannot be applied to all the problems. Therefore, Chapter 9 investigates the veto strategy and trust-based veto strategy for combining the inductive biases of all the different algorithms. The experimental results suggest that the trust-based veto strategy is a better choice for the problem in hand due to the high number of malware instances classified correctly.

5.3 Future Work

For the future work, diverse approaches from data/text mining for the feature extraction, and feature selection can be investigated using the proposed algorithms. Features extracted from the EULA may also be combined to reduce the chances of false predictions. The PUP detection problem can as well be addressed through other machine learning techniques, e.g., the rule based or neural networks using the file features presented in this thesis. This work currently predicts one kind of PUP using the proposed techniques; the detection process can be generalized for more than one class. The classification techniques, presented in all the chapters, are tested by using the real-world data in the controlled environment. These techniques are required to be tested and validated in the real-world settings. Another aspect, which needs to be addressed, is to extend the decision strategy in ensembles, especially for the trust-based veto voting. The extended trust-based veto voting strategy shall be evaluated and validated statistically. The proposed framework, in Chapter 9, is required to apply on the social network data to indicate the online potentially unwanted programs.

Six

Detection of Spyware by Mining Executable Files

Raja Khurram Shahzad, Syed Imran Haider, Niklas Lavesson

Abstract

Spyware represents a serious threat to confidentiality since it may result in loss of control over private data for computer users. This type of software might collect the data and send it to a third party without informed user consent. Traditionally, two approaches have been presented for the purpose of spyware detection: Signature-based Detection and Heuristic-based Detection. These approaches perform well against known Spyware, but have not been proven to be successful at detecting new spyware. This paper presents a Spyware detection approach by using Data Mining (DM) technologies. Our approach is inspired by DM-based malicious code detectors, which are known to work well for detecting viruses and similar software. However, this type of detector has not been investigated in terms of how well it is able to detect spyware. We extract binary features, called *n*-grams, from both spyware and legitimate software and apply five different supervised learning algorithms to train classifiers that are able to classify unknown binaries by analyzing extracted *n*-grams. The experimental results suggest that our method is successful even when the training data is scarce.

6.1 Introduction

Programs that have the potential to violate the privacy and security of a system can be labeled as Privacy Invasive Software [18]. These programs include: spyware, adware, trojans, greyware, and backdoors. They may compromise confidentiality, integrity, and availability of the system and may obtain sensitive information without informed user consent [82,83]. This information is valuable for marketing companies and also generates income for advertisers from online ads distribution through adware. This factor works as a catalyst for elevating the spyware industry [18]. Traditionally, advertisements for computer users are spread by sending spam messages, but such advertisements are not targeted toward a specific segment of users as no information about the users is available to the spammers. On the other hand, data collected by spyware may be used for customized ads spread through adware to an individual user.

Originally, viruses represented the only major malicious threats to computer users and since then much research has been carried out in order to successfully detect and remove viruses from computer systems. However, a more recent type of malicious threat is represented by spyware and this threat has not been extensively studied. According to the Department of Computer Science and Engineering at the University of Washington, spyware is defined as "*Software that gathers information about use of a computer, usually without the knowledge of the owner of the computer, and relays the information across the Internet to a third party location*" [21]. Another definition of spyware is given as "*Any software that monitors user behavior, or gathers information about the user without adequate notice, consent, or control from the user*" [18]. The major difference between the definitions involves user consent, which we regard as an important concept when it comes to understanding the difference between spyware and other malicious software.

Unlike viruses, which are always unwanted, spyware can sometimes be installed with the users expressed consent, since it may provide some useful functionality either on its own or by an accompanying software application. Due to this reason spyware overlaps the boundaries of what is considered legal and illegal software and thus falls in a grey-zone. However, in most cases, the spyware vendors do not seem to provide the user with any realistic opportunity to give an informed consent or to reject the installation of a software application in order to prevent spyware. Vendors embed spyware in regular software, which is installed with the application or by using hacking methods [84]. The installed spyware may be capable of capturing keystrokes, taking screen shots, saving authentication credentials, storing personal email addresses and web form data, and thus may obtain behavioral and personal information about users. It may also communicate system configuration including hardware and software, system accounts, location information, and information about other aspects of the system to a third party. This can lead to financial loss, as in identity theft and credit card fraud[1]. The symptoms of spyware infection vary, but spyware may, e.g., show characteristics like nonstop appearances of advertisement pop-ups, open a web site or force the user to open a web site, which has not been visited before, install browser toolbars without seeking acceptance from the user, change search results, make unexpected changes in the browser, display error messages, and so forth. Furthermore, other indications of spyware may include a noticeable change in computer performance after installation of new software, auto-opening of some piece of software or the default home page in a web-browser, a changed behavior of already installed software, the occurrence of network traffic without any request from the user, and increased disk utilization even in perceivably idle conditions [84]. Some researchers have predicted that advanced spyware can possibly take control of complete systems in the near future [85].

The awareness about spyware and its removal is considered low and outside the competence of normal users [86, 87]. Even if users have anti-virus software installed, it may not be helpful against spyware until it is designed particularly for this threat, as spyware differ from regular viruses, e.g., in that it uses a different infection technique [88].

[1]www.us-cert.gov/reading_room/spywarehome_0905.pdf [accessed 18-05-2009]

Viruses normally replicate themselves by injecting their code into executable files and spread in this way, which is not the case for most spyware. Specific anti-spyware tools have been developed as countermeasures, but there seem to be no single anti-spyware tool that can prevent all existing spyware because, without vigilant examination of a software package, the process of spyware detection has become almost impossible [89]. Current anti-spyware tools make use of signature-based methods by using specific features or unique strings extracted from binaries or heuristic-based methods by using on the basis of rules written by experts who define behavioral patterns as approaches against spyware. These approaches are often considered ineffective against new malicious code [33,88]. Moreover, since most heuristic approaches have been developed in order to detect viruses, it is not certain whether they would be capable of detecting new types of spyware because spyware use stealth systems and they do not employ any specific routines like viruses, which may be associated explicitly with spyware [88].

This paper presents a spyware detection method inspired by data mining-based malicious code detection. In this method, binary features are extracted from executable files. A feature reduction method is then used to obtain a subset of data, which is further used as a training set for automatically generating classifiers. This method is different from signature-based or heuristic-based methods since no specific matching is performed. In this method, the generated classifiers are used to classify new, previously unseen binaries as either legitimate software or spyware. In our experiments, we employ 10-fold cross-validation in order to evaluate classifiers on unseen binaries. We use accuracy and the Area under Receiver Operating Characteristic (ROC) curve as metrics for the evaluation of classifier performance.

6.2 Background

The term spyware first appeared in a Usenet post on October 16, 1995 about a piece of hardware that could be used for espionage. In 2000, the founder of Zone Labs, Gregor Freund, used the term in a press release for Zone Labs firewall product[2]. Since then, spyware has spread rapidly and several attempts to prevent this spread have been made. In 2001, the use of data mining was investigated as an approach for detecting malware [33] and this attempt attracted the attention of many researchers. Since then, several experiments have been performed to investigate the detection of traditional malicious software such as viruses, worms, and so forth, by using data mining technologies.

The objective of the aforementioned data mining experiment [33] was to detect new and unseen malicious code from available patterns. Data mining is the process of analyzing electronically stored data by automatically searching for patterns [90]. Machine Learning algorithms are commonly used to detect new patterns or relations in data, which are further used to develop a model, i.e., a classifier or a regression function. Learning algorithms have been used widely for different data mining problems to detect patterns and to find correlations between data instances and attributes. In order to represent malware instances in a suitable format for data mining purposes, many researchers have used *n*-grams or API calls as their primary type of feature. An *n*-gram is a sequence of *n* elements from a population. It can, e.g., represent a character or a word. The length of an *n*-gram can be either fixed (e.g., *uni*grams, *bi*grams, and *tri*grams) or variable. In experiments for the detection of malware, sequences of bytes extracted from the hexadecimal dump of the binary files have been represented by *n*-grams. In addition to the use of such byte sequences, some experimental studies have been conducted using data from End User License Agreements, network traffic, and honeypots.

The 2001 data mining study of malicious code [33] used three types of features, i.e., Dynamic-link Library resource information, consecutive printable characters (strings) and byte sequences. The data set consisted of 4,266 files out of which 3,265 were malicious and 1,001 were legitimate

[2]http://zonealarm.com

or benign programs. A rule induction algorithm called Ripper [91] was applied to find patterns in the DLL data. Naive Bayes (NB), a learning algorithm based on Bayesian statistics [90], was used to find patterns in the string data and n-grams of byte sequences were used as input data for the Multinomial Naive Bayes algorithm [90]. A data set partitioning was performed in which two data sets were prepared, i.e., a test data set and a training data set. This is to allow for performance testing on data that are independent from the data used to generate the classifiers. The Naive Bayes algorithm, using strings as input data, yielded the highest classification performance with an accuracy of 97.11 %. The study also implemented a signature-based algorithm and compared its results to those of the data mining algorithms. The data mining-based detection rate of new malware was twice as high in comparison to the signature-based algorithm.

Following this study, a large number of researchers [29,65,66,67,68,92,93] have devoted their efforts for encountering malicious code, which is in most of the cases either viruses or worms, by using data mining. Only two studies [88, 94] focused specifically on spyware. References [64, 65, 95] used n-grams of byte code as features while others [29, 68] used opcodes. They all were successful in having more than 97 % of accuracy. In a different study [92], an experiment was performed on network traffic filtered by network scanner, but still having suspicious malicious code. Two different types of features were used: n-grams of size 5 and Windows Portable Executable header data. This study was successful in achieving an Area under the ROC curve score of 0.983. Reference [66] performed an experiment for detection of viruses on a data set of 3,000 files. The study performed experiments on sequence lengths ranging from 3 to 8. The best result was obtained using a sequence length of 5. The results indicated that classifier performance could be increased by using shorter sequences. Reference [68] performed an experiment for the detection of Trojans. In this study, instruction sequences were used as features. The primary data set contained 4,722 files. Out of these, 3,000 files were Trojans and the rest were benign programs. A detection of compilers, common packer was also performed on data set and the feature set was also systematically reduced. Three types of algorithms were analyzed; Random Forest (RF), Bagging, and Decision Trees (DT). The study used ROC as analysis for measuring performance and the best results for false positive rate, overall accuracy and area under the ROC curve were achieved with the Random Forest classifier.

Reference [88] has replicated the work of [33] but with a focus on spyware collected in 2005. The purpose was to specifically evaluate the suitability of the technique for spyware detection. The data set consisted of 312 benign executables and 614 spyware executables. These spyware applications were not embedded (bundled) with any other executables. The Naive Bayes algorithm was evaluated, using a window size of 2 and 4, with 5-fold cross-validation. Cross-validation is a statistical method that is used to systematically divide the available data into a predetermined number of folds, or partitions [90]. Prediction models, or classifiers, are generated by applying a learning algorithm to n-1 folds and then evaluated on the nth fold. The process is repeated until all folds have been used for evaluation once. Even though criticism has been directed towards (over) belief in the cross-validation performance estimates [96], the method is still widely regarded as a reasonable and robust performance estimation method, especially when the data is scarce. The experiment showed that the overall accuracy was higher when using a window size of four.

The spyware problem is also different from that of detecting viruses or worms as vendors of spyware-hosting applications usually include them in bundles with popular free software. The End User License Agreement (EULA) may very well mention the spyware, in order for the spyware vendors to avoid legal consequences, but this information is given in a way that makes it difficult for the average user to make an informed consent. In addition, the EULAs from both legitimate software vendors and spyware vendors normally contain thousands of words, and this makes it hard for users to interpret the information. Reference [94], therefore, investigated the possibility to automatically detect spyware by mining the EULA. This study is similar to the studies carried out on spam detection by using data mining. The studied data set contained 996 EULAs out of which 9.6 % were associated with spyware. The study applied 17 learning algorithms on two data sets, represented by a bag-of-words and meta EULA model, respectively. The performances of the 17 classifiers were compared with a baseline classifier, ZeroR, which predicts the class of an instance by always assigning the majority class, e.g., the class that the majority of the instances in the training data set belong to. ZeroR is commonly used as a baseline when evaluating other learning algorithms. The results indicated that the bag-of-words model is better than the meta EULA model. Results also indicated that it is indeed possible to distinguish between legitimate software and spyware by automatically analyzing the corresponding EULAs.

A majority of the reviewed studies use *n*-grams to represent byte sequences. Except for Reference [88] and Reference [94], all the studies were performed on malware or viruses. Moreover, some studies [33, 68, 88] featured data sets with almost a double number or an equal number of malicious files than benign files. Other studies [67, 92] use a population in which a third consists of malicious files. This situation is arguably unrealistic, since in real life, the number of malicious files compared to benign files is much lower. Most of the studies used standard data sets available for malware or virus research. These data sets contain individual malicious executables. Thus, the executables are not embedded or bundled with other executables, which is the common situation for spyware. We have only been able to find one malicious code detection study that focuses on spyware [88]. This study performed experiments using *n*-grams for data representation, in particular, with $n = 2$ and $n = 4$. The latter configuration yielded the best results. However, other experiments on malicious code [92] have shown better results for $n = 5$. We, therefore, argue that a larger set of *n*-values need to be evaluated for the spyware domain.

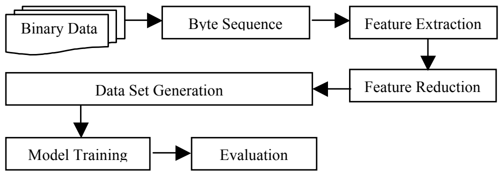


Figure 6.1: Experimental process

6.3 Proposed Method

The focus of our analysis is executable files for the Windows platform. We use the Waikato Environment for Knowledge Analysis (Weka) [97] to perform the experiments. Weka is a suite of machine learning algorithms and analysis tools, which is used in practice for solving data mining problems. First, we extract features from the binary files, and we then apply a feature reduction method in order to reduce data set complexity. Finally, we convert the reduced feature set into the Attribute-Relation File Format (ARFF). ARFF files are ASCII text files that include a set of data instances, each described by a set of features [90]. Figure 6.1. shows the steps involved in our proposed method.

6.3.1 Data Collection

Our data set consists of 137 binaries out of which 119 are benign and 18 are spyware binaries. The benign files were collected from CNET Download[3], which certifies the files to be free from spyware. The spyware files were downloaded from the links provided by SpywareGuide.com[4], which hosts information about different types of spyware and other types of malicious software. The rather low amount of gathered spyware is attributed to the fact that most of the links provided by SpywareGuide.com were broken, i.e., these links did not lead to pages where the spyware executables could be downloaded. We have yet to find, or build a larger spyware data set.

[3]http://download.com
[4]http://spywareguide.com

6.3.2 Malicious File Percentage

Reference [29] has shown that for their particular study, the MFP needed to be equal to, or lower than 15 % of the total population in order to yield a high prediction performance. Relating to our data set, the MFP is almost 14 %. However, it is important to stress that we have yet to uncover evidence to support that the recommended MFP leads to improved results in the general case.

6.3.3 Byte Sequence Generation

We have opted to use byte sequences as data set features in our experiment. These byte sequences represent fragments of machine code from an executable file. We use xxd[5], which is a UNIX-based utility for generating hexadecimal dumps of the binary files. From these hexadecimal dumps, we may then extract byte sequences, in terms of *n*-grams of different sizes.

6.3.4 *n*-gram Size

A number of research studies have shown that the best results are gained by using an *n*-gram size of 5 [66, 92]. In the light of previous research, we chose to evaluate three different *n*-gram sizes (namely: 4, 5, and 6) for the experiments.

6.3.5 Parsing

We first extract byte sequences of the desired *n*-size. Each row contains one *n*-gram and the length of a single row is thus equal to the size of *n*.

6.3.6 Feature Extraction

The output from the parsing is further subjected to feature extraction. We extract the features by using two different approaches: the Common

[5]http://linux.about.com/library/cmd/blcmdl1_xxd.htm

Feature-based Extraction (CFBE) and the Frequency-based Feature Extraction (FBFE). The purpose of employing two approaches is to evaluate two different techniques that use different types of data representation, i.e., the occurrence of a feature and the frequency of a feature. Both methods are used to obtain Reduced Feature Sets (RFSs), which are then used to generate the ARFF files.

Common Feature-based Extraction

In CFBE, the common *n*-grams (byte sequences) are extracted from the binary files, one class at a time.

Frequency-based Feature Extraction

The word frequency can be defined in various ways. In statistics, it basically represents the number of occurrences or repetitions of some observation at a specific time or from some specific category. In our study, the word frequency means the number of occurrences of some specific *n*-gram in a certain class or the number of repetitions of some specific *n*-gram in a particular class. In FBFE, all the *n*-grams were sorted and the frequency of each *n*-gram in each class is calculated. All *n*-grams, within a specified frequency range, are extracted and the rest is discarded. In the frequency calculation, we discovered that there were a few uninteresting *n*-grams, e.g., 0x0000000000, 0xFFFFFFFFFF, and 0x0000000001. Even though these instances were few (less than 10), their frequencies were high (more than 10,000). Thus, the frequency analysis helped us to define three suitable frequency ranges: 1-49, 50-80, and 81-500. The number of *n*-grams in the 50-80 frequency range tends to be almost equal to the number of *n*-grams in the 81-500 range.

6.3.7 Feature Reduction

Features were reduced in both the CBFE and FBFE method. In CBFE, the common features gained from all files were sorted. Only one representation of each feature was considered in one class. CBFE has produced a better reduced feature set. For example, the reduced feature set for *n*

Table 6.1: Feature Statistics

Size / Features	***n* = 4**	***n* = 5**	***n* = 6**
Benign Features	26,474,673	21,179,768	17,649,809
Spyware Features	8,357,458	6,685,971	5,571,645
Total Features	34,832,131	27,865,739	23,221,454
FR = 1 - 49	26,269,292	21,987,533	18,746,618
FR = 50 - 80	5,282	3,226	2,286
FR = 81 - 500	6,018	3,929	2,788
CFBE	536	514	322

= 4 contains only 536 features compared to 34,832,131 for the complete set. In FBFE, the frequency of each *n*-gram is calculated. Reduced features were obtained with three frequency ranges 1-49, 50-80, and 81-500. After analysis of the number of *n*-grams in each frequency range, it was decided that the 1-49 frequency range will not be included in the experiments, since the number of *n*-grams even in the reduced feature set was too high. For example, for an *n*-gram size of 5, the total number of *n*-grams was 27,865,739 and the number of *n*-grams included in the reduced feature set for the 1-49 frequency range was 21,987,533, which indicated the presence of a large amount of uninteresting features. Table 6.1. shows some statistics regarding the number of features in each class, the total number of features, and the reduced feature sets, based on different frequency ranges for both FBFE and CFBE.

6.3.8 ARFF Generation

Two ARFF databases based on frequency, and common features were generated. All input attributes in the data set are represented by Booleans, i.e., either a certain *n*-gram or the *n*-grams within a certain frequency range are represented by either 1 or 0 (present or absent).

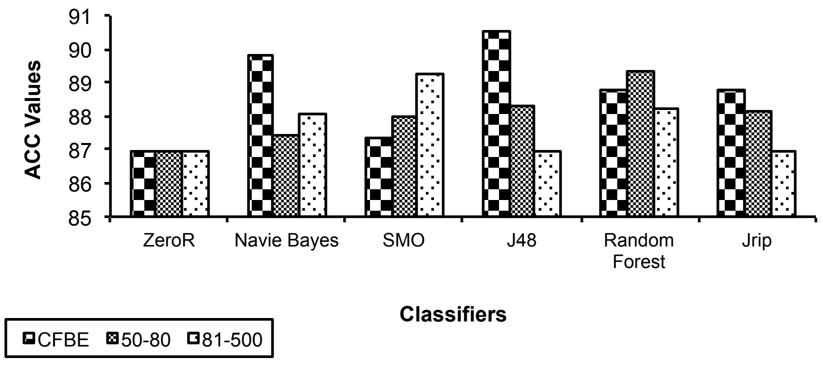


Figure 6.2: Comparison of Accuracy with $n = 6$

6.3.9 Classifiers

Previous studies are not conclusive about which learning algorithm generates the best classifiers for problems similar to the studied problem. However, the results have provided us with a basis for choosing ZeroR, Naive Bayes, Support Vector Machine (SVM) algorithm, i.e., Sequential Minimal Optimization (SMO), C4.5 Decision Tree (J48), Random Forest and JRip as candidates for our study. ZeroR is used only as a baseline for comparison. For our purpose, it can be viewed as a random guesser, modeling a user who makes an uninformed decision about a piece of software. A Naive Bayes classifier is a probabilistic classifier based on Bayes theorem with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. This of course, is seldom true for real-life applications. Nevertheless, the algorithm has shown good performance for a wide variety of complex problems. SVMs, which are used for classification and regression, work by finding the optimal hyperplane, which maximizes the distance/margin between two classes. J48 is a decision tree-based learning algorithm. During classification, it adopts a top-down approach and traverses a tree for classification of any instance. Moreover, Random Forest is an ensemble learner. In this ensemble, a collection of decision trees are generated to obtain a model that may give better predictions than a single decision tree. Meanwhile, JRip is a rule-based learning algorithm.

6.3.10 Performance Evaluation Criteria

We evaluate each learning algorithm by performing cross-validation tests to ensure that the generated classifiers are not tested on the training data. From the response of the classifiers, the relevant confusion matrices were created. Four metrics define the elements of the matrix: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). TP represents the correctly identified benign programs while FP represents the incorrectly classified spyware programs. Correspondingly, TN represents the correctly identified spyware programs and FN represents the wrongly identified benign programs. The performance of each classifier was evaluated using the AUC metric and the (overall) Accuracy (ACC) metric. The later is defined in Equation (6.1). AUC is essentially a single point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [90]. Therefore, it plots the True Positive Rate (TPR, see Equation (6.1)) on the x-axis in function of the False Positive Rate (FPR, see Equation (6.2)) on the y-axis at different points. TPR is the ratio of correctly identified benign programs while FPR is the ratio of wrongly identified Spyware programs. ACC is the percentage of correctly identified programs. In many situations, ACC can be a reasonable estimator of performance (the performance on yet unseen data). However, AUC has the benefits of being independent of class distribution and cost [98]. In many real world problems, the classes are not equally distributed and the cost of misclassifying one class may be different to that of misclassifying another. For such problems, the ACC metric is not a good measure of performance, but it may be used as a complementary metric.

$$TPR = \frac{TP}{TP + FN} \tag{6.1}$$

$$FPR = \frac{FP}{TN + FP} \tag{6.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

6.4 Results

Table 6.2. and Table 6.3. show the results for each *n*-gram size for both the CFBE and the FBFE method. Two feature sets were produced as a result of the FBFE approach. The first feature set includes instances from the frequency range of 50-80 and the second set includes instances from the 81-500 frequency range. Each table shows the results of the baseline classifier and five different learning algorithms. As stated earlier, we represent classification performance using two metrics: AUC and ACC. Algorithms that are significantly better or worse than the baseline in terms of AUC according to the corrected paired t-test (confidence 0.05, two-tailed) are also indicated. It is not the main objective of this study to determine the optimal algorithm or the highest possible performance. Hence, we did not tune the parameters of the learning algorithms, i.e., all algorithms are default configured.

6.4.1 Results for *n* = 4

Using the feature set produced by the CFBE feature selection method for $n = 4$, the J48 decision tree classifier achieves the highest accuracy results. However, it only performs slightly better than the Random Forest model and the JRip classifier. In comparison, the accuracy of Naive Bayes is mediocre while the support vector machines classifier (SMO) achieved the lowest accuracy. In summary, all included algorithms have performed better than the baseline. When comparing the AUC-based performance results, the Random Forests model achieved the highest score while J48 performed at a mediocre level.

Table 6.2: Accuracy results of experiment

n	Method	ZeroR	NaiveBayes	SMO	J48	Random Forest	JRip
4	CBFE	86.92 (2.72)	88.20 (6.17)	86.56 (8.13)	89.89(5.10)	89.48(5.52)	89.45(4.94)
	FR 50_80	86.92 (2.72)	83.70 (8.20)	88.52 (7.53)	88.52(5.89)	87.07(6.47)	88.07(6.28)
	FR 81_500	86.92 (2.72)	79.37 (12.28)	89.65 (5.93)	88.41(6.21)	88.82(5.66)	85.84(7.31)
5	CBFE	86.92 (2.72)	89.36 (5.43)	83.41 (8.59)	89.88(4.78)	88.61(6.19)	89.15(4.36)
	FR 50_80	86.92 (2.72)	87.25 (6.19)	86.39 (6.65)	87.15(7.72)	88.45(5.93)	88.21(5.85)
	FR 81_500	86.92 (2.72)	87.39 (5.69)	88.42 (5.96)	84.68(6.97)	87.94(5.57)	85.95(7.17)
6	CBFE	86.92 (2.72)	89.80 (4.89)	87.37 (6.98)	90.54(4.33)	88.74(5.75)	88.77(5.40)
	FR 50_80	86.92 (2.72)	87.41 (6.26)	88.02 (6.10)	88.30(5.51)	89.30(5.45)	88.15(4.21)
	FR 81_500	86.92 (2.72)	88.04 (6.13)	89.29 (7.26)	86.96(7.50)	88.25(6.01)	86.98(7.37)

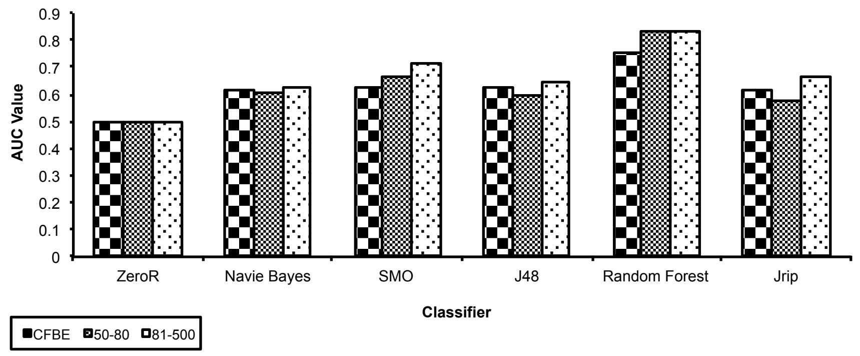


Figure 6.3: Comparison of Area under ROC Curve with n = 6

In the 50-80 frequency range, SMO and J48 produced the highest ACC results. Moreover, in the higher range (81-500), SMO yields the highest ACC-based performance. In the frequency range of 50-80, SMO and J48 are slightly better than JRip and in the frequency range of 81-500; SMO is performing slightly better than J48. When comparing results across these two frequency ranges, it is obvious that the difference in accuracy is negligible.

6.4.2 Results for n = 5

Similarly to the results for n = 4, the feature set produced by the CFBE feature selection method was suitable as a data representation for the J48 algorithm, which again yielded the best ACC results. In comparison, the SMO produced the worst ACC result of all included algorithms for this particular data set. In fact, the results of the SMO are even lower than the baseline. In contrast, the difference between the results of J48, Naive Bayes and JRip is small and all algorithms performed better than the baseline. In terms of the AUC, the Random Forest algorithm was the best performer, followed by the J48. For the 50-80 frequency range, the Random Forest algorithm yielded the highest ACC results. However, it only slightly outperformed the JRip rule inducer. J48 performed mediocre at this range. Moreover, in the 81-500 range, the SMO yielded the highest ACC results while NB and RF were mediocre. In terms of the area under the ROC curve, the Random Forest algorithm outperformed the other algorithms for both the 50-80 and the 81-500 range. When comparing results obtained for both frequency results, the 50-80 frequency range seems to be more suitable on average than the 81-500 frequency range.

6.4.3 Results for $n = 6$

The data sets generated for $n = 6$ proved to be most successful in terms of both accuracy and the area under the ROC curve. The feature set produced by the CFBE feature selection algorithm was used in conjunction with the J48 decision tree algorithm to yield a top accuracy score of 90.5 %. It slightly outperformed the NB algorithm, which was followed by RF and JRip. The support vector machines algorithm yielded the lowest ACC score. All algorithms performed better than the baseline. For the AUC Random Forest was the best performer, yielding a top AUC score of 0.83.

6.5 Discussion

The feature sets generated by the CFBE feature selection method generally produced better results with regard to accuracy than the feature sets generated by the FBFE feature selection method. However, the reverse situation seems to be true when AUC is used as an evaluation metric. Overall, the results suggest that the two higher frequency ranges are more suitable than the lowest. The best AUC result was obtained by using the Random Forest algorithm, an n-gram size of 6, and the highest frequency range. However, the 50-80 frequency range yielded better AUC results than the other frequency ranges when the size of n-grams was 4. When comparing the performance on the data sets generated by the FBFE method, it is clear that the ACC results for the 50-80 frequency

Table 6.3: Area Under ROC Curve results of experiment

n	Method	ZeroR	Naive Bayes	SMO	J48	Random Forest	JRip
	CBFE	0.50(0.00)	0.61(0.17)*	0.63(0.18)*	0.63(0.16)*	0.73(0.23)	0.61(0.15)
4	FR 50_80	0.50(0.00)	0.62(0.20)	0.71(0.18)	0.62(0.16)	0.78(0.20)	0.60(0.16)
	FR 81_500	0.50(0.00)	0.62(0.23)	0.67(0.19)*	0.61(0.17)	0.78(0.20)*	0.56(0.14)
	CBFE	0.50(0.00)	0.62(0.17)*	0.58(0.16)	0.63(0.15)*	0.68(0.22)*	0.59(0.13)*
5	FR 50_80	0.50(0.00)	0.61(0.17)	0.66(0.17)*	0.64(0.20)*	0.76(0.23)*	0.61(0.16)*
	FR 81_500	0.50(0.00)	0.61(0.16)*	0.67(0.17)*	0.62(0.24)	0.78(0.21)*	0.66(0.19)*
	CBFE	0.50(0.00)	0.61(0.19)	0.62(0.17)	0.63(0.16)	0.75(0.22)	0.61(0.16)
6	FR 50_80	0.50(0.00)	0.60(0.16)	0.66(0.17)*	0.59(0.15)	0.82(0.17)*	0.57(0.12)
	FR 81_500	0.50(0.00)	0.62(0.18)*	0.71(0.20)*	0.65(0.18)*	0.83(0.17)*	0.66(0.18)*

range are better than those for the 81-500 frequency range. This can easily be viewed in Figure 6.2. Meanwhile, the AUC results are very close to each other for both ranges as shown in Figure 6.3. Consequently, more experiments, e.g., with larger amounts of data and a wider variety of learning algorithms are needed in order to fully understand which data representation and feature selection method is optimal for our purpose.

6.6 Conclusions and Future Work

Data mining-based malicious code detectors have been proven to be successful in detecting clearly malicious code, e.g., like viruses and worms. Results from different studies have indicated that data mining techniques perform better than traditional techniques against malicious code. However, spyware has not received the same attention from researchers, but it is spreading rapidly on both home and business computers. The main objective of this study was, therefore, to determine whether spyware could be successfully detected by classifiers generated from n-gram based data sets, which is a common data mining-based detection method for viruses and other malicious code.

In order to find a suitable data representation, we analyzed n-gram-based byte sequences of different sizes from a range centered on $n = 5$, which has proven to be an appropriate value that yields high performance for similar experiments. We then evaluated five common learning algorithms by generating classifiers and using 10-fold cross-validation and the corrected paired t-test. Moreover, two different feature selection methods were compared for all algorithms and n-gram sizes. Since no suitable spyware data set was available, we collected spyware and legitimate binaries and generated a small data set for the purpose of validating our approach. The experiments indicate that the approach is successful, achieving a 90.5 % overall accuracy with the J48 decision tree algorithm when using $n = 6$ and the common n-gram feature selection method. The success of the approach is also indicated by an AUC score of 0.83 with the Random Forest algorithm when using $n = 6$ and the frequency-based feature selection method. Currently, the false positive rate is quite high for most combinations of algorithms and data sets. However, we believe that one of the primary reasons for this is that the data set is small. In particular, the number of spyware files is too low. In data mining, it is believed that larger set of data can produce better results [90]. So a larger set of data can be tested to have better classification with higher ACC and lower false positive rate. With regard to AUC, which is our primary evaluation metric, all algorithms were statistically significantly better than the baseline, but with different combinations of *n*-gram size and feature selection method. Thus, from our experiments, we can conclude that it is possible to detect spyware by using automatically generated classifiers to identify patterns in executable files.

We hope that data mining techniques can help the researcher community and security experts to label the software or the home user to have an informed decision before installation of any software. For future work, we plan to gather a larger collection of binary files, especially spyware binaries as no standard data set of spyware is currently available, and to evaluate our approach when the data set features represent opcodes instead of arbitrary bytes. Additionally, we aim to develop a hybrid spyware identification method that is based on a combination of EULA-based and executable-based detection techniques.

Seven

Detecting Scareware by Mining Variable Length Instruction Sequences

Raja Khurram Shahzad, Niklas Lavesson

Abstract

Scareware is a recent type of malicious software that may pose financial and privacy-related threats to novice users. Traditional countermeasures, such as anti-virus software, require regular updates and often lack the capability of detecting novel (unseen) instances. This paper presents a scareware detection method that is based on the application of machine learning algorithms to learn patterns in extracted variable length opcode sequences derived from instruction sequences of binary files. The patterns are then used to classify software as legitimate or scareware, but they may also reveal interpretable behavior that is unique to either type of software. We have obtained a large number of real world scareware applications and designed a data set with 550 scareware instances and 250 benign instances. The experimental results show that several common data mining algorithms are able to generate accurate models from the data set. The Random Forest algorithm is shown to outperform the other algorithms in the experiment. Essentially, our study shows that, even though the differences between scareware and legitimate software are subtler than between, say, viruses and legitimate software, the same type of machine learning approach can be used in both of these dissimilar cases.

7.1 Introduction

This paper addresses the problem of detecting scareware, i.e., scam software with different forms of malicious payloads[1], and presents a machine learning-based approach for detection of this type of software. Many reports have been published in the media regarding malicious software (malware) such as viruses and worms. Such reports have arguably increased the awareness of novice computer users about basic security issues. Consequently, users are becoming more conscious about the security and privacy of their systems and data. Through media, friends, colleagues and security experts, users have been advised to install detection software such as anti-virus software or other protective measures. Success stories about the detection and mitigation of virus and worm threats have probably also played a part in enhancing the general security awareness. However, personal computers are becoming more and more attractive targets for cyber criminals due in part to the electronic financial transactions that are nowadays performed by private citizens from their personal computers. Notable examples of such transactions include Internet bank sessions and online credit card payments. This development has caused a shift in the focus of the malware authors from self-spreading malware such as worms, which are easily detectable due to their distribution techniques, to privacy invasive malware [99].

A recent addition to the family of privacy invasive malware is known as rogue software, rogueware, or scareware. In the remainder of this paper, the latter term will be used to denote this type of software. Scareware represents scam applications that usually masquerade as security applications such as anti-malware software or more specifically anti-

[1]http://www.microsoft.com/security/antivirus/rogue.aspx

virus software. In reality, however, scareware provides a minimum level of security or no security at all[2,3]. This type of software is especially crafted to include fake scanning dialogs, artificial progress bars and fake alerts [100]. Scareware may display fake lists of infected files and sometimes such lists are so unsophisticatedly generated that they include files that may not even exist on the computer or they may be incompatible with the operating system [34]. Figure 7.1. shows the fake scanning dialog of a particular scareware Rouge:W32/Winwebsec[4]. The fake scanning processes and the fake results are of course used to scare users into believing that their system has been compromised and that it is infected with malicious content. The fake presentations are essentially carried out to convince the user that they need anti-virus software or some other form of protection. As a remedy for the fake situation, scareware offers a free download, which may also be bundled with other malware (such as trojans) or it may facilitate the installation of additional malware. Scareware may also trick users into paying registration fees in order to scan their system more thoroughly to remove (fake) warnings. Such an example is shown in Figure 7.2., which is the screen-shot of payment screen displayed by Rouge:W32/Winwebsec[4]. The additional malware, which has been installed instead of protective software, remains on the targeted computer regardless of whether the registration fee is actually paid or not. Such additional malware is typically used to collect personal data of the user or to launch different forms of attacks.

7.1.1 Background

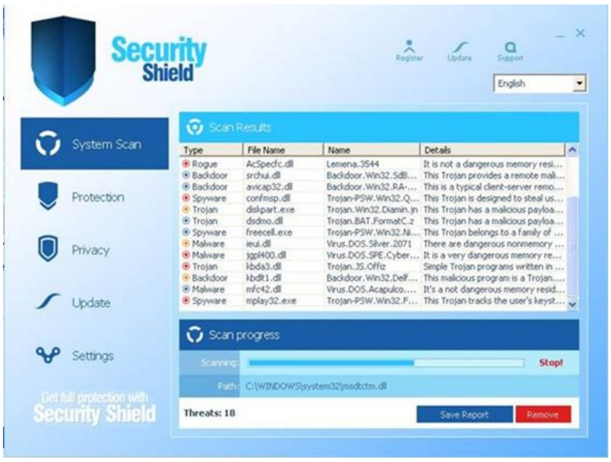
In 2003, Secure Works observed that spam advertisements for fake anti-virus software were being sent to users by the utilization of a vulnerability in the Microsoft Messenger Service[5]. Two years later, in 2005, Microsoft reported about the presence of scareware on web sites and

[2]http://voices.washingtonpost.com/securityfix/2009/03/obscene_profits_fuel_-rogue_ant.html

[3]http://www.lavasoft.com/

[4]http://winsec.se/?cat=166

[5]http://www.secureworks.com/research/threats/popup-spam/



Type	File Name	Name	Details
Rogue	AcSpecfc.dll	Lemena.3544	It is not a dangerous memory resi...
Backdoor	srchui.dll	Backdoor.Win32.SdB...	This Trojan provides a remote mali...
Backdoor	avicap32.dll	Backdoor.Win32.RA-...	This is a typical client-server remo...
Spyware	confmsp.dll	Trojan-PSW.Win32.Q...	This Trojan is designed to steal us...
Trojan	diskpart.exe	Trojan.Win32.Diamin.jn	This Trojan has a malicious payloa...
Trojan	dsdmo.dll	Trojan.BAT.FormatC.z	This Trojan has a malicious payloa...
Spyware	freecell.exe	Trojan-PSW.Win32.Ni...	This Trojan belongs to a family of ...
Malware	ieui.dll	Virus.DOS.Silver.2071	There are dangerous nonmemory ...
Malware	jgpl400.dll	Virus.DOS.SPE.Cyber...	It is a very dangerous memory re...
Trojan	kbda3.dll	Trojan.JS.Offiz	Simple Trojan programs written in ...
Backdoor	kbdlt1.dll	Backdoor.Win32.Delf...	This malicious program is a Trojan....
Malware	mfc42.dll	Virus.DOS.Acapulco....	It's a not dangerous memory resid...
Spyware	mplay32.exe	Trojan-PSW.Win32.F...	This Trojan tracks the user's keyst...

Scanning:

Path: C:\WINDOWS\system32\msdtctm.dll

Threats: 18

Figure 7.1: Scanning screenshot of Rouge:W32/Winwebsec[4]

web servers. Since then and arguably due to the overwhelming financial incentive to malware authors, the scareware has been increasing. The scareware distribution mechanism is different from other malware (such as viruses or worms). Scareware reaches the target machine by employment of social engineering, stealth techniques, or both of these approaches. User interaction is required when scareware is distributed through social engineering. For this purpose, advertisements are either sent via spam e-mail or posted on popular social networking web sites. Scareware is misleadingly marketed as legitimate software and, with user interaction; it is downloaded and installed on personal computers. When it comes to stealth techniques, vulnerabilities in web browsers or other popular software are exploited in order to employ a so-called drive-by download mechanism. Essentially, scareware is downloaded and installed without any user interaction using such a mechanism. It has been reported that the monetary conversion rate of the fees obtained for fake scanning services can be as much as 1.36 %, which can result in a gross income of \$21,000 - \$35000 for a period of 44 days [99]. Panda Labs reported that an approximate overall gross income of 34 million per month is generated by scareware [101]. Late in 2009, Symantec Corporation reported about 43 million installation attempts from more than 240 distinct families of scareware[6]. Recently a Swedish newspaper Aftonbladet[7] reported that according to U.S Department of Justice, 9, 60,000 users were victims of rouge, which caused a loss of 460 million krona. This alarming situation has received the attention of legitimate security software companies. CA Global Security Advisor[8], Secure Works[9] and Microsoft[10] published advisories about scareware, which describe the general functionality of scareware and tips for identifying this type of software [34]. To reduce the probability of being fooled by scareware, novice users are advised to install legitimate anti-malware software. However, the problem with such software is that users need to update it on regular basis as novel families of scareware are continuously appearing.

7.1.2 Traditional Countermeasures

Current anti-malware programs primarily rely on either signature-based methods or heuristic-based methods for the detection of scareware; techniques that were originally developed for detecting computer viruses. The signature-based approach revolves around the use of signature databases that contain byte strings that are unique to different instances of software. If these databases are allowed to become more than a couple of weeks old, the detection rate will be significantly reduced due to the fact that the approach cannot detect recent scareware for which it lacks recorded signatures [34]. The second approach, which relies on heuristic-based methods, is based on more general rules that, e.g., may define malicious or benign behavior. For both methods, anti-malware vendors need to catch novel instances, analyze them, create new signatures or rules and then update their databases. It is safe to say that,

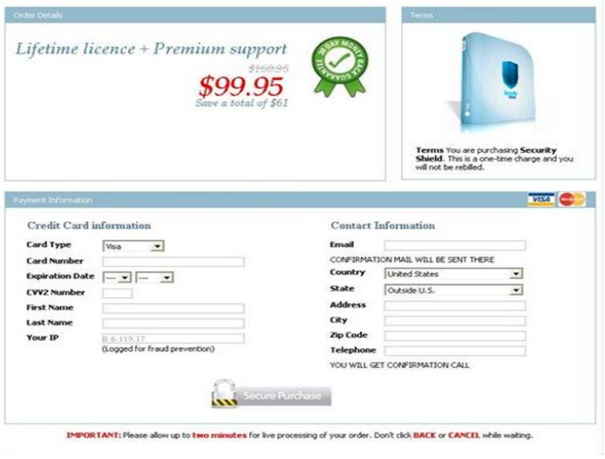
[6]http://www.symantec.com/about/news/resources/press_-kits/detail.jsp?pkid=istr_rogue_security

[7]http://www.aftonbladet.se/nyheter/article13218796.ab

[8]http://cainternetsecurity.net

[9]http://www.secureworks.com/

[10]http://www.microsoft.com



Order Details

Lifetime licence + Premium support

~~$160.95~~

$99.95

Save a total of $61

Terms

Terms You are purchasing **Security Shield**. This is a one-time charge and you will not be rebilled.

Payment Information

Credit Card information

Card Type	Visa
Card Number	
Expiration Date	--- / ---
CVV2 Number	
First Name	
Last Name	
Your IP	8.6.119.17 (Logged for fraud prevention)

Contact Information

Email	
	CONFIRMATION MAIL WILL BE SENT THERE
Country	United States
State	Outside U.S.
Address	
City	
Zip Code	
Telephone	
	YOU WILL GET CONFIRMATION CALL

IMPORTANT: Please allow up to **two minutes** for live processing of your order. Don't click **BACK** or **CANCEL** while waiting.

Figure 7.2: Payment screenshot of Rouge:W32/Winwebsec[4]

between database updates, users may be exposed to novel scareware instances. Thus, it is important for users to have up-to-date and reliable protective measures.

7.1.3 Scope and Aim

In this paper, we present results from an experimental evaluation of a new scareware detection method. The aim of this method is to extend the traditional heuristic detection approach by employing machine learning. The objectives of the study are to assess the performance of the proposed method, which can be described as an automated system for extracting typical behavior of scareware and benign software in the shape of variable length instruction sequences, and to analyze such fragments of behavior in order to improve upon the existing knowledge about scareware detection.

7.1.4 Outline

The remainder of this paper is organized as follows. Section 7.2 presents related work by first introducing necessary concepts and terminology in

Section 7.2.1 and then reviewing related studies in Section 7.2.2. Section 7.3 then describes the employed methodology and the data preprocessing steps. Section 7.4 reviews the experimental procedure. The subsequent sections present the experimental results and the analysis. Finally, Section 7.7 concludes the paper and gives some pointers to future work.

7.2 Related Work

7.2.1 Concepts and Terminology

To overcome the deficiency of traditional techniques concerning the detection of novel instances, the potential of various approaches, such as agent-based technologies and artificial neural networks have been investigated. Data mining (DM) and machine learning (ML) methods have been extensively investigated in the field of text classification and have showed promising results for many applications. As we shall see, it is possible to benefit from this area of research when addressing the scareware detection problem. However, the idea of using DM and ML methods for making the malware detection process automated and for extending the heuristic-based detection approach for traditional malware is not new; it originates from a study conducted in 2001 [33].

The process of ML-based malware classification essentially follows standard classification and can thus be divided into two sub stages: training and testing. During the training stage, classifiers are generated from training sets that feature some type of extracted malware and benign file information as well as the predetermined classification of each file and the predictive performance of the classifiers is then evaluated during the testing stage. For malware classification, data sets have been prepared using various representations of files and by using different features that are either present in the files or obtained from any kind of meta analysis (for example, runtime generated digital footprints). Features that are commonly extracted from a binary file include: byte code *n*-grams, printable strings and instruction sequences. The *n*-gram is a sequence of n characters or n extracted words. Other features that are present in binary files and that may also be used include system calls (to application programming interfaces). The use of opcodes as an alternative form of representation has also been suggested [102]. An assembly instruction contains an operation code (opcode) and maybe one or more operands for performing the operation. Opcodes or sequences of opcodes may be represented using *n*-grams, which, in turn, can be viewed upon as words or terms if the learning problem is defined as a text categorization problem.

In text categorization, text files are commonly represented using the bag of words model, which is based on Salton's vector space model [103]. A vocabulary of words or terms is extracted from the so-called document set. For each term (*t*) in the vocabulary, its frequency (*f*) in a single document (*d*) and in the entire set (D) is calculated. A weight is assigned to each term, usually equal to its *f* in *d*; such weights are denoted term frequencies (*tf*). When the frequency (F) of each term is calculated in D, this is called Document Frequency (DF). The *tf* value of a term is further divided by the frequency of the most frequent term in the document, i.e., max(*tf*) to obtain a normalized Term Frequency (TF) within the range of [0-1] as shown in Equation (7.1). An extended version of TF-DF is TF Inverse Document Frequency (TF-IDF), which combines TF and DF as shown in Equation (7.2); where N is the number of documents in the entire data set and DF is number of *d* in which *t* appears.

$$TermFrequency = \frac{tf}{max(tf)} \tag{7.1}$$

$$TFInverseDocumentFrequency = TF \times \log[\frac{N}{DF}] \tag{7.2}$$

The problem of *n*-gram-based malware classification in this context is perhaps different from the general text categorization case since a huge vocabulary or very large feature sets have to be produced. The size of the vocabulary creates two problems: most ML algorithms cannot directly process the vocabulary and a vast number of terms in the vocabulary do not provide any valuable information for classification. Therefore, it is necessary to obtain a subset of features by applying feature selection. The Categorical Proportional Difference (CPD) algorithm is a rather recent example of such an algorithm. In a number of text categorization experiments, CPD has outperformed other traditional feature selection algorithms such as: chi-square, information gain, mutual information, and odds ratio [104]. CPD represents a measure of the degree to which a word contributes in discriminating a specific class from other classes [104]. The possible outcome of CPD falls between [-1. . . 1] where a CPD value close to -1 indicates that a word occurs in an equal number of instances in all classes and a value of 1 or in proximity to 1 indicates that a word occurs only in one class. A is the number of times word *w* and class *c* occur together and let B the number of times word *w* occurs without class *c*, then we may define CPD for a particular word *w* and class *c* as shown in Equation (7.3):

$$CPD(w, c) = \frac{A - B}{A + B} \tag{7.3}$$

The reduced feature set can now be converted, e.g., into the Attribute-Relation File Format (ARFF). ARFF files are structured ASCII text files that include a set of data instances, each described by a set of features [38]. ARFF files are used as input to the Waikato Environment for Knowledge Analysis (Weka) [97] before applying learning algorithms in order to build and analyze classifiers. Essentially, Weka is a suite of machine learning algorithms and analysis tools for solving or analyzing data mining problems. There are, of course, many alternatives to Weka, but we argue that this workbench is particularly fitting for developing our approach since it is released as open source and may be tuned, extended, or changed in any way.

7.2.2 Related Directions of Research

Opcodes have already been used to build signature databases that can be searched to detect different variants of worms [69]. To avoid the problem of having to manually update the databases of the scanners, data mining algorithms were later used as part of a scientific study to build a generic scanner [93]. In this study, experiments were performed on two different data sets: the first data set contained the opcode of each instruction and the second data set contained the opcode as well as the first operand of each instruction. The frequency of appearance in the virus class and in the benign class was used as a basis for feature selection. Results showed that the first data set produced better results than the second. In another study, opcode *n*-grams of different sizes were constructed to detect novel malware. By addressing the class imbalance problem properly, an accuracy of 96 % was achieved [29]. The idea of using variable length instruction sequences was conceived as part of an attempt to detect worms. Frequently occurring instruction sequences were analyzed using ensemble learners to classify novel instances of worms. In an attempt to detect a more recent type of malware, called spyware, hexadecimal *n*-grams were used to represent binary files [20]. The most common *n*-grams for each class together with overall high frequency *n*-grams were used as features for building the classifiers. The spyware detection rate was recorded to be 90.5 %.

Hexadecimal *n*-grams have been used extensively as features in traditional malware classification problems. Experiments have been performed on viruses, worms and trojans. These types of malware are typically very distinct from the standard benign software program. Moreover, only a few studies have used only the opcode from the instruction as the feature of choice [29,69,93]. Today, very little is known about the appropriateness of using opcodes or instruction sequences as features when trying to detect the type of malware that is more similar to benign software in terms of behavior. In this paper, we investigate the concept of scareware which, to the best of our knowledge, has not been investigated in terms of how well it can be detected by mining instruction sequences.

7.3 Methodology

Generalizing the scareware detection method so it can detect novel instances can arguably be regarded as quite important for user protection. Another problem regarding the detection of scareware is that it may resemble legitimate software to such extents that it is difficult to detect differences. Recently, data mining classification algorithms have been heavily applied in order to automate and extend the heuristic-based methods for detection of traditional malware. It is, therefore, of interest to investigate how well such classification algorithms can detect scareware. Consequently, we present a static analysis method based on data mining, which extends the general heuristic detection approach. In this context, a dynamic analysis method is used to detect malware instances by investigating runtime footprints while static analysis is carried out on files without any runtime execution. Our data set contains Windows-based executable scareware and benign files and this choice was made since the Windows operating system is still the most commonly used operating system, especially for novice users, and it is often considered more vulnerable than, e.g., Unix-based operating systems. We have disassembled our initial file database into instruction sequences and then we extracted the opcodes from each instruction. The extracted opcodes were combined into ordered lists, instruction sequences (IS), to produce our vocabulary. Each word in vocabulary is of variable length. We have used TF-IDF and CPD for generating the final data sets.

7.3.1 File Sampling and Data Set Design

As the threat of scareware is relatively new compared to, say, viruses and worms, there is unfortunately no default or public data set available for researchers to build classification models from. Therefore, we have created a data set of 800 files out of which 550 are scareware (provided by Lavasoft[3] from their scareware collection). The remaining 250 files are benign and were downloaded from the web site CNET Download[11]. This web site claims that the software provided is spyware free. How-

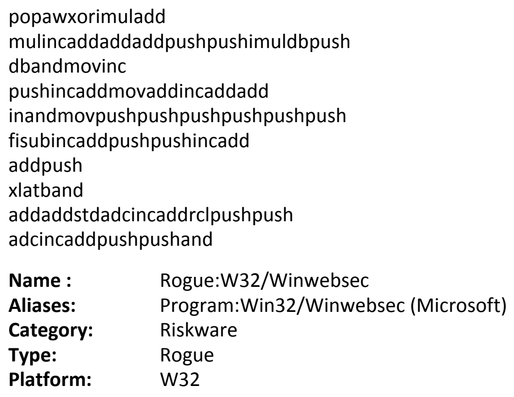
[11]http://download.com

ever, after downloading software from the web site and scanning it with a commercial version of the F-Secure Client Security software[12], we discovered that some files were actually infected by so-called riskware. The infected instances were removed from the data set.

7.3.2 Extraction and Data Preparation

For the purpose of our experiment, we needed to convert our data set to a format that could be processed by learning algorithms. We decided to represent files by using extracted instruction sequences as features. The advantage of using IS as a primary feature is that IS represent program control flow blocks, which cannot be presented by binary or hexadecimal *n*-grams or printable strings. Moreover, each IS in this study represents a function that can be located within the actual program for the purpose of deeper analysis, even though such a step is out of scope in the presented paper. We disassembled each program using the Netwide disassembler[13] (Ndisasm) , which was configured to auto-synchronous mode to avoid misalignment between the data and code segments. The generated output, from all the file segments, was stored in regular text files and each entry contains the memory address of the instruction field as well as the opcode and the operands. The disassembled files were further processed through a parser to obtain the instruction sequences (ordered lists of opcodes). During the extraction process, the end of an instruction sequence was determined by identifying a conditional or unconditional control transfer instruction or function boundary. It is worth noting that these identified control transfer instructions (such as: call, iret, jmp or jnz) were not included in the generated instruction sequences. In this way, we obtained variable length instruction sequences. Each row in output contains single IS. Figure 7.3. shows the instruction sequences extracted from a scareware Rouge:W32/Winwebsec[4] alongwith some other related information of this particular scareware.

[12]http://www.f-secure.com/
[13]http://nasm.us/



```
popawxorimuladd
mulincaddaddaddpushpushimuldbpush
dbandmovinc
pushincaddmovaddincaddadd
inandmovpushpushpushpushpushpush
fisubincaddpushpushincadd
addpush
xlatband
addaddstdadcincaddrclpushpush
adcincaddpushpushand
```

Name :	Rogue:W32/Winwebsec
Aliases:	Program:Win32/Winwebsec (Microsoft)
Category:	Riskware
Type:	Rogue
Platform:	W32

Figure 7.3: Instruction Sequence Extracted from Rouge:W32/Winwebsec

7.3.3 Feature Selection

Feature selection is performed to measure the correlation of each feature with its class (scareware or benign). It is also performed to estimate the role of that specific feature in classification task. The measures used for feature selection by any feature selection methods are not biased to any classification algorithm or class, which helps us in comparing the performances of different classification algorithm.

Our disassembled files were in text format and each file can be read as text string so we decided to use the bag of words model, since it has been proven to be a suitable model for similar problems. We used the *StringToWordVector* filter in Weka to parse each string, extract the vocabulary and produce word vectors. For our experiment, each word represents a unique IS. We used TF-IDF for the weight calculation of each word. Our vocabulary features top 1,625 unique words. We decided to perform a secondary feature selection to eliminate features, which will not contribute significant in classification task. We applied CPD to obtain reduced feature sets. As it is difficult to know beforehand the optimal number of features to remove, we decided to generate a number of data sets where each set was generated by keeping a different number of attributes. This process resulted in 19 reduced data sets for which 5-95 %

of the original features were kept.

7.4 Experiment

The aim of the experiment is to evaluate classifier performance on the task of detecting scareware by learning from variable length instruction sequences and to assess the impact of feature selection using categorical proportional difference. Learning algorithms can be categorized according to their learning bias, that is, by the way their designs restrict the search space and dictate how this space is traversed. When categorizing the commonly used algorithms in this manner, a rather small number of algorithm families can be identified, e.g., tree inducers, rule set inducers, neural networks, instance based learners, and Bayesian learners. We have tried to select at least one representative algorithm from each family. As our study extends the heuristic based detection technique that uses rules set, so we used families of algorithms that either uses rules or help in developing rule set. These families of algorithms are rules based and decision tree. Except these families we also used support vector machine, Bayesian theorem based algorithms and nearest neighbor concepts for classification. All the algorithms were used at their default configuration in Weka.

7.4.1 Learning algorithms

ZeroR

ZeroR is a rule-based algorithm. ZeroR works as a random guesser, modeling a user that makes an uninformed decision about software by always predicting the majority class (the class to which most of the data instances belong) [38]. This algorithm is frequently used as a baseline to measure the performance gain of other algorithms in classification against chance.

JRip

JRip is an implementation of the Ripper algorithm. This algorithm tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances they cover) and accuracy [105]. A data instance is classified as positive if a rule matches; otherwise it is classified as negative. JRip also features an optimization step in which redundant or bad rules are discarded.

J48

J48 is a decision tree based learning algorithm, which uses the concept of information entropy [106]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed. For partitioning, J48 uses the attribute with the highest information gain and stops if all instances of same class are present in the subset. In learning, they adopt top-down approach and traverse the tree to make a set of rules, which is used for classification.

Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) belongs to support vector machines. During classification, SMO finds the optimal hyper-plane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [107].

Naive Bayes

Naive Bayes (NB) is based on Bayes theorem and generates a probabilistic classifier with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. Therefore, presence (or absence) of a particular feature of a class is not dependent on the presence (or absence) of any other feature [108].

IB*k*

IB*k* is *k*-nearest neighbor classifier, which uses Euclidean distance [109]. Predictions from the neighbors is obtained and weighted according to their distance from test instance. Majority class of closest *k* neighbors is assigned to new instance.

Random Forest

Random Forest (RF) is an ensemble learner. A specified number of decision trees are created and their mode is obtained for prediction predictions [110]. Being an ensemble learner, it has superiority of having combined decision, which is not the case for other algorithms, therefore, it is expected to produce better accuracy than single decision tree.

7.4.2 Evaluation

We tested each learning algorithm by performing 10 fold cross-validation (CV) tests to ensure that the generated classifiers are not tested on the training data. Confusion matrices were generated by using the responses from classifiers. The following four measures defined the elements of the generated confusion matrices: True Positives (TP) represent the correctly identified scareware programs, False Positives (FP) represent legitimate software that has been classified as scareware, True Negatives (TN) represent correctly identified legitimate programs and False Negatives (FN) represent scareware programs that were incorrectly classified as legitimate software applications. We argue that the false negatives carry the highest cost from the users' perspective.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified scareware, as shown in Equation (7.4). False Negative Rate, which is the percentage of wrongly identified malicious programs (see Equation (7.5)), and Accuracy (ACC), the percentage of correctly identified programs (see Equation (7.6)). The last evaluation measure used was Area Under Receiver Operating Characteristic Curve (AUC). AUC is essentially a single-point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [38]. Therefore, it plots the DR on the x-axis in function of the False Positive Rate (FPR) on the y-axis at different points. FPR is the percentage of wrongly identified benign programs. The higher AUC of an algorithm indicates that this algorithm is more robust and better in classification. In many situations, accuracy can also be a reasonable estimator of performance (the performance on completely new data). However, AUC has the benefits of being independent of class distribution and cost [98] unless the skewness of the class distribution is extreme.

$$DetectionRate = \frac{TP}{TP + FN} \tag{7.4}$$

$$FalseNegativeRate = \frac{FN}{TP + FN} \tag{7.5}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.6}$$

7.5 Results

The main experimental results, that is, the AUC of the seven included algorithms on the 19 data sets, are shown in Table 7.1. In this table, ZeroR is used as a baseline (and can be regarded as a random guesser) with AUC of 0.500 (0.000) for all the data sets. All algorithms had performed better than base algorithm. Random Forest outperformed the other algorithms and its best performance (DR of 0.977, FNR of 0.023 and FPR of 0.197) was recorded at the 60 % keep level (a data set with 974 features). The Naive Bayes yielded an acceptable detection rate (0.857), but its FPR at different data sets was too high (i.e., up to 0.688) for practical use. Moreover, Naive Bayes also exhibited a high variance in performance related to the different data sets. Due to this behavior, it is not possible to consider this algorithm as reliable for the studied problem. On the same data set other algorithms also achieved either the highest AUC or near to the highest value with ignorable differences such as SMO achieved

Table 7.1: Learning algorithms AUC results for different levels of feature selection

Data	**ZeroR**	**SMO**	**Naive Bayes**	**IB*k***	**Jrip**	**J48**	**Random Forest**
5%	0.500(0.000)	0.717(0.119)	0.787(0.030)	0.778(0.030)	0.657(0.129)	0.500(0.000)	0.781(0.030)
10%	0.500(0.000)	0.812(0.038)	0.829(0.044)	0.851(0.036)	0.809(0.040)	0.788(0.031)	0.857(0.036)
15%	0.500(0.000)	0.860(0.045)	0.802(0.049)	0.896(0.037)	0.817(0.054)	0.869(0.043)	0.937(0.027)
20%	0.500(0.000)	0.878(0.040)	0.814(0.045)	0.928(0.036)	0.868(0.047)	0.877(0.048)	0.958(0.021)
25%	0.500(0.000)	0.864(0.042)	0.809(0.045)	0.924(0.034)	0.872(0.044)	0.874(0.054)	0.959(0.020)
30%	0.500(0.000)	0.883(0.041)	0.804(0.045)	0.908(0.043)	0.885(0.038)	0.876(0.056)	0.960(0.021)
35%	0.500(0.000)	0.885(0.041)	0.805(0.045)	0.927(0.036)	0.883(0.044)	0.880(0.053)	0.962(0.022)
40%	0.500(0.000)	0.880(0.043)	0.815(0.045)	0.938(0.032)	0.885(0.041)	0.901(0.043)	0.964(0.018)
45%	0.500(0.000)	0.892(0.041)	0.832(0.044)	0.930(0.034)	0.887(0.043)	0.904(0.047)	0.965(0.021)
50%	0.500(0.000)	0.900(0.036)	0.855(0.043)	0.932(0.031)	0.893(0.043)	0.896(0.048)	0.966(0.019)
55%	0.500(0.000)	0.906(0.035)	0.897(0.040)	0.928(0.033)	0.893(0.041)	0.896(0.051)	0.969(0.020)
60%	**0.500(0.000)**	**0.910(0.033)**	**0.923(0.033)**	**0.935(0.029)**	**0.894(0.047)**	**0.900(0.044)**	**0.972(0.017)**
65%	0.500(0.000)	0.910(0.031)	0.879(0.056)	0.938(0.028)	0.893(0.042)	0.894(0.046)	0.972(0.017)
70%	0.500(0.000)	0.909(0.031)	0.710(0.056)	0.938(0.028)	0.901(0.038)	0.893(0.047)	0.970(0.019)
75%	0.500(0.000)	0.909(0.031)	0.667(0.044)	0.938(0.028)	0.898(0.039)	0.893(0.047)	0.968(0.021)
80%	0.500(0.000)	0.909(0.031)	0.657(0.043)	0.938(0.028)	0.901(0.039)	0.916(0.041)	0.970(0.021)
85%	0.500(0.000)	0.909(0.031)	0.656(0.043)	0.938(0.029)	0.898(0.043)	0.915(0.037)	0.971(0.018)
90%	0.500(0.000)	0.911(0.031)	0.658(0.042)	0.939(0.027)	0.896(0.039)	0.910(0.036)	0.970(0.019)
95%	0.500(0.000)	0.915(0.031)	0.668(0.044)	0.938(0.028)	0.901(0.039)	0.906(0.038)	0.971(0.022)

```
int 0x21
push sp
push word 0x7369
and [bx+si+0x72],dh
outsw
```

Figure 7.4: A disassembled function in one particular scareware instance

AUC 0.910 and highest AUC with 95 % features was 0.915, IB*k* achieved 0.935 while the highest AUC was 0.938 for dataset of 65 % features. JRip and J48 both achieved AUC of 0.894 and 0.900 respectively while their highest AUC was 0.901 and 0.916 for data sets with 70 % and 80 % features. Due to these ignorable minor differences in results, we considered that data set with 60 % features is a better option for our problem.

7.6 Analysis

We created 19 different data sets and each data set was having 5 % less features than its successor. Experimental results indicated that a step of 5 % was not enough to create significant difference in the result. NB has been an exception to this, which showed a random trend with increased or decreased percentage of features. However, if we look at the data sets created with the difference of 10 % features then the difference in results is quite prominent. If the step is increased up to 20 % difference of features, then a clear and understandable difference of results is present. If we review the overall performances on the various data sets, it is clear that the performance of most algorithms was quite high on the 60 % feature selection level. It seems that number of kept features at this level is properly balanced with the number of instances from each class.

In order to understand the classification process and to find interesting features, we analyzed the models generated by JRip, J48, and SMO. Models created by other algorithms cannot be visualized, so it was not possible to perform their analysis. We found three kinds of features i.e., features present only in scareware, features present only in legitimate, features which were treated differently by different algorithms. Table 7.2. shows some selected features with a high impact on the classification decision. Features 1 and 7 are used to indicate scareware by all three models. However, features 2, 5, 8, and 9 were considered as a scareware indicative feature by two algorithms, but were ignored by the remaining algorithms. Features 2, 3, 6, and 10 seem to be considered as legitimate software indicative features by all algorithms. Finally, feature 4 is regarded as a legitimate indicator by JRip, but as a scareware indicator by SMO.

In order to demonstrate the information provided by a single feature, we traced the features from Table 7.2. to the disassembled binary files. One such example is provided in Figure 7.4. As per our understanding, the function in Figure 7.4. seems to indicate an attempt to transfer some specific string data to the user for display or transfer from user to some other end. The particular contents of the memory are not available to us since we are performing a static analysis and thus to get a deeper understanding, we would have to manually analyze a larger portion of disassembled code. However, it is clear that some functionality is present only in scareware instances, which would suggest that it is possible to differentiate them from benign files on a general level. However, it would be hard for a human expert to detect and analyze such subtle differences; therefore, we argue that our automatic approach is superior, especially when considering the fact that regular applications can contain several thousands of lines of code.

7.7 Conclusions and Future work

We have extended the heuristic-based detection technique using a variable length instruction sequence mining approach for the purpose of scareware detection. Since scareware is a rather recent software security threat, there are no publicly available data sets to generate classification models from. We have, therefore, obtained a large sample of scareware applications and designed an algorithm for extracting in-

Table 7.2: Selected features and their number of occurrence in each class

		Jrip		**J48**		**SMO**	
F.No	**Feature**	**S**	**L**	**S**	**L**	**S**	**L**
1	pushpushandoutsw	>0		>0		-0.1420	
2	ormovadd			>0		-0.0813	
3	inswpopaw		>=1				0.1940
4	incoutswoutsb		>=1			-0.0195	
5	addmovmovmovcmp			>0		-0.0615	
6	leadb		>1		>0		0.2016
7	dbdecmov	>0		>0		-0.0848	
8	outswarplfs			>0		-0.0223	
9	movpushmov	>0				-0.0572	
10	pushaddpush		>1				0.1704

struction sequences from these applications (and similarly for legitimate software). The data sets used in this study will be publicly available at `http://www.bth.se/com/rks`. The experimental results are promising: the Random Forest algorithm managed to yield an AUC score of 0.972 after the complete data set was processed using the categorical proportional difference feature selection algorithm. Moreover, the results also indicate that our method is trustworthy since the false negative rate (the rate of scareware classified as legitimate) is considerably low (0.023). For future work, we aim to conduct further experiments on an even larger collection of scareware and benign files. We also plan to employ a hybrid identification method, which would integrate variable length instruction sequences with features extracted from, e.g., the end user license agreement or the information about the system calls a particular program makes.

Eight

Accurate Adware Detection using Opcode Sequence Extraction

Raja Khurram Shahzad, Niklas Lavesson, Henric Johnson

Abstract

Adware represents a possible threat to the security and privacy of computer users. Traditional signature-based and heuristic-based methods have not been proven to be successful at detecting this type of software. This paper presents an adware detection approach based on the application of data mining on disassembled code. The main contributions of the paper is a large publicly available adware data set, an accurate adware detection algorithm, and an extensive empirical evaluation of several candidate machine learning techniques that can be used in conjunction with the algorithm. We have extracted sequences of opcodes from adware and benign software and we have then applied feature selection, using different configurations, to obtain 63 data sets. Six data mining algorithms have been evaluated on these data sets in order to find an efficient and accurate detector. Our experimental results show that the proposed approach can be used to accurately detect both novel and known adware instances even though the binary difference between adware and legitimate software is usually small.

8.1 Introduction

The aim of this study is to investigate adware detection and to develop an algorithm that accurately detects known and unknown adware instances. Adware may be defined as software that is installed on the client machine with the objective of displaying ads for the user of that machine [22]. Basic adware may also be bundled with extra functionality or software to invade the privacy of a user by monitoring his or her surfing activities or preferences in order to display related pop-up or pop-under advertisements. However, advanced adware may, for example: read data from locally stored files, collect surfing or chat related information, and even create remote connections for transferring and installing software in the future by making a system vulnerable and compromised [22]. These and similar capabilities may even turn the adware into a spyware or some other type of malicious software (malware). Arguably, adware compromises the confidentiality, and in some cases also the integrity and availability of computer systems. Analogously to computer viruses, which infect executable computer files, adware may be installed automatically when the user visits infected websites, installs freeware or shareware or when the user tries to open infected E-mail attachments [111]. The presence of adware is often mentioned in the End User License Agreement (EULA), but in a manner, which makes it difficult for the average users to comprehend or even notice [94]. As a consequence, the user's informed consent is thus not obtained. Because of its presence in the EULA, adware vendors often claim that their software should be regarded as benign [94]. Such claims along with the differences in policies and regulations decided upon by different countries place adware in a grey zone in terms of legal status.

The adware problem is growing continuously due to the profound monetary gains for adware developers [22, 112]. The users' awareness about adware and its potential consequences is generally considered to be low [111]. Currently, the major commercial anti-virus tools try to detect instances of adware by relying on static or dynamic analysis such as signature-based and heuristic approaches (which were developed for detection of viruses). These techniques have a deficiency in detecting unknown or new instances and can be bypassed in different ways [29]. Two popular commercial tools for adware detection are SpyBot[1] and AdAware[2], which rely on signature based approach. Hence they require frequent update of their signature database and can detect only known instances.

Consequently, in this paper, we present a (static) detection method based on data mining. We have proposed an automated means for extracting instruction sequences (ISes) from adware and benign files in order to capture the behavior of the corresponding software. Our method extracts the operation code (opcode) from each instruction and then produces a data set in which each instance is described by sequences of opcodes. As the remainder of this paper will show, our approach is feasible for detecting adware despite the fact that the binary files of this type of software are sometimes quite similar to legitimate software.

8.1.1 Aim and Scope

In this paper, we present the results from an experimental study of adware detection. The aim is to determine the success of using data mining techniques for the detection of unseen and new instances of adware. Additionally, we investigate the relationship between opcode *n*-gram size and the number of features required to generate accurate detection models. Our hypothesis is that: it is possible to find a balance between the size of *n*-grams (that are used to represent opcode sequences) and the number of features (the number of *n*-grams) that yields a model of reasonable classification performance.

[1]http://www.safer-networking.org/
[2]http://lavasoft.com

8.2 Background

Adware is different from other malware since it may be installed with or without the consent of the user [23]. Users may accept its presence knowingly for using freeware software or unknowingly when it is obfuscated in the EULA. The user may also be fooled into installing adware when trying to install other software or the installation of adware may be carried out as a background task without any human interaction at all [23]. Thus, it is important to be able to automatically detect adware. As mentioned earlier, traditional detection techniques, i.e., signature-based and heuristic methods have a deficiency in detecting novel instances of traditional malware, spyware and adware. In the signature-based technique, specific features or unique strings are extracted from binaries, which are later used for detection of malware. However, a copy of the malware is required to extract and develop a signature for detection purposes. Due to this fact, signature-based techniques are usually not capable of detecting novel (unseen) instances of malware. In the heuristic technique, human experts define rules for detecting behavioral patterns for malware detection. This technique is capable of detecting novel instances albeit with limited capacity and may be prone to false alarms.

8.2.1 Data Mining-based Detection

To overcome the aforementioned deficiency in detection techniques, Machine Learning (ML) methods, as an alternative approach, were applied for malware detection in 2001 [33]. Since then, different studies have been conducted for detection of traditional malware such as viruses, worms, and so forth, by applying ML and Data Mining (DM) technologies. DM helps in analyzing the data, with automated statistical analysis techniques, by identifying meaningful patterns or correlations. The results from this analysis can be summarized into useful information and can be used for prediction [38]. ML algorithms are used for detecting patterns or relations in data, which are further used to develop a classifier or a regression function [38].

For DM purposes, researchers have prepared their experimental data sets either by using different representations of binary files or by extracting a certain type of features that is present in the files. A binary file may be converted into hexadecimal code, binary code or ISes as a means for representation. Moreover, these representations may be further used to create *n*-grams, which are fixed-size strings. Other features that are commonly present in files are printable text strings or calls to an application-programming interface. The use of opcodes as an alternative representation has also been suggested in [102]. An opcode is a part of the instruction for an operation in machine language. It may or may not include one or more operands for performing an operation such as an arithmetical operation or transferring program control.

When the data set is prepared for machine learning classification tasks, a class imbalance problem may arise. Typically, the imbalance problem occurs in a data set when one class has significantly more instances in comparison to another class or other classes. Due to this problem, the generated classifier tends to misclassify instances of the least represented class(es) and thus the problem may result in degradation of classification performance. Therefore, it is necessary to address the imbalance problem during data set preparation. One approach is of course to try to ensure that all classes are equally represented. This approach, however, turns out to be practically impossible to adopt in many real world problems since there usually is a great shortage of data instances of certain classes.

8.2.2 Feature Selection

Another important task when preparing the data set is to reduce the data set complexity while maintaining or improving performance of the classification model. For this purpose, the most common approach is to apply a feature selection algorithm. The objective of feature selection is basically to apply a feature quality measure to prioritize the available features and then keep only the best features from the prioritized list. In the information retrieval domain, the bag-of-words model (in which the logical order of words has no importance) performs better than other models in representing text documents [113].

Different feature selection measures, such as Document Frequency, Gain Ratio, and Fisher Score, are commonly used for obtaining reduced data sets [38]. Categorical Proportional Difference (CPD) is a relatively new addition in the feature selection algorithm family for text classification tasks [104]. The experiments have shown that CPD outperforms common feature selection methods such as chi-square and information gain. CPD represents a measure of the degree to which a word contributes to differentiating a specific class from others [104]. The possible value of CPD is within the interval of -1 and 1. A CPD value close to -1 indicates that a word to large extent occurs in an equal number of instances in all classes and a value in proximity of 1 indicates that a word occurs only in one class. Given that A is the number of times word w and class c occur together and B is the number of times word w occurs without class c, then we may define CPD for a particular word w and class c as follows (see Equation (8.1)):

$$CPD(w,c) = \frac{A - B}{A + B} \tag{8.1}$$

The reduced feature sets can then be used for data mining purposes and can be used as input to learning algorithms. Many types of learning algorithms are available. Therefore, it is important to choose suitable algorithms with respect to the problem at hand.

8.3 Related Work

Due to legal issues and lawsuits from adware vendors, anti-virus vendors are hesitant to classify any software as adware [112]. Therefore, we have not been able to find any specific approaches for detecting adware. However, we argue that it is important to detect adware to let users exercise the right to make an informed choice about the software they install. In previous work, opcodes have been used for detection of different variants of worms and some types of spyware [69]. From the original malware, opcodes were extracted and paired with labels. With these pairs, researchers developed signatures, which were matched with pairs of variants of malware. A three-stage scanning was performed, which was successful in detecting the different variants. In another study, an attempt to detect unknown malware was made by extracting opcodes from malware and then converting them into sequences of opcodes [29]. In their experiment, the researchers applied three classifiers out of which two were boosted and achieved 93 per cent accuracy. In yet another study, variable length instruction sequences were used as a representation for the detection of worms. This time, researchers applied Bagging and were successful in achieving 96 per cent accuracy [71]. In an attempt to detect spyware, *n*-grams of hexadecimal representation were used as features [20]. This attempt was successful in obtaining 90.5 % accuracy. Most of the reviewed detection experiments on traditional malware were performed using hexadecimal *n*-grams as features. Only a few researchers seem to have considered opcodes as features and then only from the code segment of the studied files [29, 71]. The files in these experiments were disassembled using commercial dissemblers to obtain the IS. Moreover, most of these studies have not considered the class imbalance problem, which may lead to unnecessarily high rates of misclassification. In conclusion, most of the work concerning malware detection focuses on viruses, worms, and trojans. It is not clear whether the same type of detection methods would be successful when dealing with adware, which is more similar to legitimate software than such types of malware. Nevertheless, adware represents a serious threat to privacy and, as such, the research on adware is important, especially in terms of detection approaches.

8.4 Method

We propose a static DM-based analysis method, which includes disassembling the adware and benign files during preprocessing. We aim to evaluate our proposed method for detecting unknown and new instances as well as existing instances of adware.

8.4.1 Overview

The focus of our analysis is Windows-based executable files, since the Windows operating system has been considered to be more vulnerable to adware as opposed to, say, Unix-based operating systems. When any software is disassembled, the generated output contains text, which may represent hexadecimal dumps, binary dumps or ISes. We argue that text categorization techniques can, therefore, be applied on disassembled output to distinguish between adware and benign software. Thus, we disassemble executable files to obtain ISes and then extract opcodes from those instruction sequences. The extracted opcodes are converted into a vocabulary data set. Each word in the vocabulary data set is an *n*-gram of a specific size, which represents a feature. Although the size of each word in a particular vocabulary set is fixed, the length is variable. For example, if we observe a data set where the *n*-gram size is 4 then each word is constructed by joining four opcodes, where each opcode may have a different length. We use Term Frequency - Inverse Document Frequency (*tf-idf*) to measure the significance of every word in order to extract significant features. The generated data is converted into the Attribute-Relation File Format (ARFF) data set file format. The ARFF files are further processed with CPD to obtain feature-reduced data sets, which are used as input to the Waikato Environment for Knowledge Analysis (Weka) [97] to perform the classification experiments. Weka is a suite that includes a large set of machine learning algorithms as well as analysis tools for solving data mining problems.

8.4.2 Data Set Generation

No public data set is available for use in adware detection experiments as opposed to what is available for, e.g., virus and intrusion detection. Therefore, we have created a data set with 600 files out of which 300 files are adware and 300 files are considered as benign. All files represent executable binaries for the Windows operating system. The benign files stem from two sources: a copy of the Windows XP operating system was installed on a clean computer to obtain benign files, e.g., small programs such as notepad, paint, clock, and so forth. Second, to represent files available on the Internet, programs were downloaded from CNET Download[3]. This website claims to provide spyware free software; however, when downloaded data set was scanned with a commercial version of the F-Secure Client Security software[4], some instances were found infected by so-called riskware. The infected instances were replaced by other benign files. Adware files were obtained from a malware database[5].

8.4.3 File Size and Data Size Analysis

File size and data size analysis has to be performed to investigate potential imbalance problems. When adware and benign programs were collected, it was observed that the mean file size of the two software program groups was quite different. Therefore, it was necessary to avoid an unbalanced number of instructions since different file sizes may produce a varying number of ISes. This may further lead to a class based difference in the generated vocabulary, which in turn may lead to an imbalance problem. Therefore, we decided to restrict the maximum file size to 512 KB for this particular study. It was also considered that the total number of files and the total size of these files should be approximately equal in both data sets.

8.4.4 Disassembly and Opcode Extraction

The collected programs were disassembled to get instruction sequences in assembly Language. This step was performed using the Netwide disassembler[6] (Ndisasm), which is commonly available for UNIX/Linux operating systems. Ndisasm disassembles binary files without under-

[3]http://download.com
[4]http://f-secure.com
[5]http://lavasoft.com
[6]http://http://www.nasm.us/

standing (correctly processing) object file formats. The generated output contained the memory address of the instruction, the byte-based location in the file and the instruction itself, i.e., the combination of opcode and operands. An application was further developed to extract the opcodes from the disassembled file. We did not just include the opcodes from the code segment of the files, but instead used opcodes extracted from any segment in the file.

8.4.5 Parsing and *n*-Gram Size

The extracted opcode data were processed further with a parser that tokenized the data to produce vocabulary/words as per a selected *n*-gram size. In a previous research study, an *n*-gram size of 4 or 5 yielded promising results for the hexadecimal representation [66,92]. In another study, an *n*-gram size of 2 for opcode representation yielded the best performance [29]. Therefore, we decided to use *n*-grams of sizes ranging from 2 to 8 while considering 4 and 5 as intermediary values. The purpose of selecting this range was to evaluate *n*-gram sizes in proximity of what has been considered adequate settings in previous research. We created seven master data sets using these *n*-gram sizes. Each row in these data sets represented one word, which is an *n*-gram of a specific size. Thus we obtained features of *n*-grams with seven different *n*-gram sizes. These data sets contain the features with different number of occurrences in each class. We also calculated the number of unique features in one class. Table 8.1. presents the vocabulary statistics for each class and data set.

8.4.6 Feature Selection

The main objective of our particular feature selection step was to obtain sets of features with a different amount of data that represents both adware and benign programs. The output obtained from the previous steps, contains huge vocabularies, which may lead to two problems, i.e., ML algorithms may not process this huge vocabulary and all words in vocabulary do not provide valuable information for classification. We

Table 8.1: Vocabulary Statistics

	Adware		**Benign**		**Final**
n	***Total***	***Unique***	***Total***	***Unique***	**(*tf-idf*)**
2	4497344	35666	4381315	25921	1236
3	2998173	452915	2920818	228780	1340
4	2248586	876451	2190581	440580	1413
5	1798843	881768	1752439	565536	1518
6	1499012	804138	1460335	630851	1630
7	1284845	727570	1251705	656345	1676
8	1124215	660092	1095219	643148	1753

used *tf-idf* for initial feature selection. The frequency n is number of times a word (or *n*-gram in our case) appears in single document, *dj*. It is not feasible to use this frequency as basis for selection of words as documents may be of different length so that some words will be more frequent regardless of their actual importance. For normalization purposes, we use Term Frequency (*tf*), which gives a measure of the importance of a word (also known as term) in document *dj*. This measure is obtained by dividing the frequency of a word, *ni,j*, with the sum of all frequencies of all words in the document *dj*. For obtaining the general importance of word in a document set, D, we use Inverse Document Frequency, *idf*. For obtaining *idf*, D is divided by the number of documents that include that particular word and then the logarithm of that value is taken. To get the final measure of a word and filter out common words, we use Term Frequency - Inverse Document Frequency. The *tf-idf* of a word is obtained by multiplying *tf* and *idf* of that particular word. By using *tf-idf*, we obtained the final data sets. The total number of final words in every data set varies. In information retrieval, it is common to use a predefined number of words obtained from *tf-idf* (such as the top 1,000 words for both classes or each class). We argue that our problem is different from normal text classification in terms that when the *n*-gram size increases, the number of unique words in each class is also increased (see Table 8.1). Therefore, we let the number of selected words depend on the data set in question instead of a predefined number. There is an additional benefit derived from this feature selection step: suppose a file in the benign data set may be infected with a zero-day threat or that the features extracted from some files are really part of the data segment instead of the code or images. For these cases, the corresponding features will be ignored due to their absence in other files of the same class.

Moreover, the output from the previous step was further processed using the CPD algorithm to create the final data sets. CPD has shown promising results in text classification, but has not been used previously for malware classification. We expected that the use of CPD would lead to better detection performance than other common feature selection methods. As the exact percentage of features to keep in order to yield optimal performance is not known beforehand, we chose to discard features in increments of 10 per cent for every generated data set. Nine final data sets for each *n*-gram size were created. These data sets can be downloaded from `http://www.bth.se/com/rks`.

8.4.7 Data mining algorithms

Previous studies on similar problems are not conclusive regarding which learning algorithm generates the most accurate classifiers. In a number of studies of malware detection, Ripper (JRip), C4.5 Decision Tree (J48), Support Vector Machines (SMO), and Naive Bayes (NB) performed better than other algorithms. In a previous study of text categorization [23], *k*-nearest neighbor (IB*k*) outperformed NB and other algorithms. Based on previous research, we selected these algorithms as candidates and compared them against ZeroR as a baseline.

ZeroR

ZeroR is a simple, deterministic rule-based algorithm. ZeroR resembles as a random guesser, which could be used to model a user that makes an uninformed decision about software by always predicting the majority

Table 8.2: Area Under ROC Curve value for *n*-gram size of 4

***n*-Size = 4**	**Naive Bayes**	**SMO**	**IB*k***	**J48**	**JRip**
10%	0.820(0.059)	0.815(0.058)	0.884(0.047)	0.824(0.057)	0.817(0.061)
20%	0.752(0.066)	0.848(0.043)	0.886(0.043)	0.832(0.058)	0.818(0.064)
30%	0.879(0.046)	0.906(0.036)	0.920(0.032)	0.884(0.045)	0.889(0.048)
40%	0.901(0.039)	0.927(0.033)	0.926(0.033)	0.896(0.053)	0.906(0.036)
50%	0.892(0.036)	0.945(0.031)	0.934(0.031)	0.886(0.049)	0.903(0.045)
60%	0.863(0.042)	0.942(0.031)	0.945(0.024)	0.888(0.049)	0.906(0.041)
70%	0.838(0.042)	0.939(0.031)	0.949(0.024)	0.885(0.045)	0.901(0.036)
80%	0.828(0.044)	0.945(0.026)	0.935(0.029)	0.884(0.044)	0.898(0.043)

class [38]. This algorithm is frequently used as a baseline to measure the performance gain of other algorithms in classification against chance.

JRip

JRip is an implementation of the Ripper algorithm [105], which tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances that are matched) and accuracy. Ripper includes intermediate and post pruning techniques to get increase the accuracy of the final rule set.

J48

J48 is a decision tree induction algorithm, extended from the ID3 algorithm, which uses the concept of information entropy [106]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed.

SMO

SMO is an implementation of the support vector machines (SVM) algorithm using Platts sequential minimization optimization. During classification, SMO tries to find the optimal hyperplane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [107]. SMO has been generalized in order to be applicable for problems in which there are more classes than two.

Naive Bayes

Naive Bayes is based on Bayes theorem and generates a probabilistic classifier with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other [108]. Clearly, such an assumption is violated in most real-world data sets. Nevertheless, the Naive Bayes algorithm has proven to generate quite accurate classifiers for many problems.

IB*k*

IB*k* is an implementation of the *k*-nearest neighbor (*k*NN) algorithm, which computes the Euclidean distance between the instance to be classified and the instances included in the training set. Predictions from the neighbors is obtained and weighted according to their distance from the test instance. The majority class of the closest *k* neighbors is assigned to the new instance [109].

8.5 Evaluation Metrics

We evaluated each learning algorithm by performing cross-validation tests. Confusion matrices were generated by using the responses from classifiers. The following four estimates defined the elements of such a matrix: True Positives (TP) represent the correctly identified adware programs. False Positives (FP) represent the incorrectly classified benign programs. True Negatives (TN) represent the correctly identified benign programs and False Negatives (FN) represent the incorrectly identified adware programs.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified adware. False Alarm Rate (FAR), which is the percentage of wrongly identified benign programs and Accuracy (ACC), the percentage of correctly identified programs. We argue that, for our problem, False Negative Rate, which is the percentage of incorrectly identified adware programs, is more important than FAR. The last evaluation parameter was Area Under Receiver Operating Characteristic Curve (AUC). AUC is essentially a single point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [38]. Therefore, it plots the DR on the x-axis in function of the FAR on the y-axis at different points. In many situations, ACC can be a reasonable estimator of predictive performance. However, AUC has the benefits of being independent of class distribution and cost [98]. In many real-world problems, the classes are not equally distributed and the cost of misclassifying one class may be different to that of misclassifying another. For such problems, the ACC metric is not a good measure of performance, but it may be used as a complementary metric.

8.6 Experimental Procedure

To investigate our hypothesis that it is possible to find a suitable combination of *n*-gram size and the number of features to yield a model of reasonable classification performance, a comprehensive set of evaluation runs was designed. Our experiment used seven different sizes of *n*-grams to create data sets and for each specific n there were nine sub sets ranging from 10 % features to 90 % features. In total, we conducted 630 10-fold cross-validation (CV) tests for each classifier, which resulted in 3,780 runs in total. Default configurations were used for all algorithms. We used corrected paired t-test (confidence 0.05, two tailed) to compare each classifier with the base line classifier ZeroR.

8.7 Results

Most of the algorithms performed well when using an *n*-gram size of 4 and the 70 % features data set. The results of all algorithms were compared with the results of ZeroR, which achieves an AUC score of 0.50 (random guessing). Figure 8.1. shows the comparison of all algorithms in terms of AUC score for *n*-gram size of 4 with 70 % features data set. The AUC scores for IB*k* for aforementioned data set are presented in Table 8.2. Considering AUC as the primary performance metric, the results clearly show that our proposed methodology is successful in detecting novel (unseen) instances of adware. IB*k* achieved the best result (AUC = 0.949, FNR = 0.022 and FAR = 0.115 with *n*=4 and 70 % attributes kept). In terms of FNR on the 70 % data set for different *n*-gram sizes, most of the algorithms achieved the highest FNR at *n*-gram size of 2. NB has shown high variance among all *n*-gram sizes for FNR and FAR with highest FNR value of 0.475 for *n*-gram size 2 and highest FAR value of

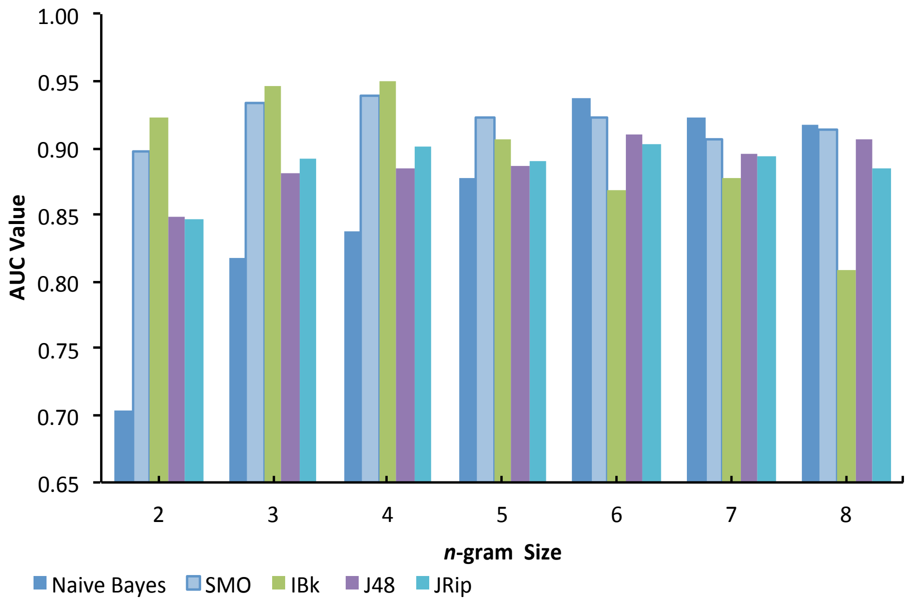


Figure 8.1: AUC of 70 % data set for all *n*-gram sizes

0.335 for *n*-gram size of 3. All other algorithms gave their highest FNR and FAR for *n*-gram size of 2. The IB*k* achieved highest FAR, i.e., 0.462 for *n*-gram size of 8.

8.8 Analysis

The results clearly show the possibility to detect adware using data mining on disassembled code and thus strengthen the validity of our hypothesis. The aim of this study is twofold. Firstly, we need to evaluate our methodology for detection and secondly, we need to find a suitable combination of *n*-gram size with percentage of features. Results have shown that adware could be detected using *n*-grams of opcodes. We have used opcode sizes ranging from 2 to 8. For our experiments, we have not considered an *n*-gram size of 1 since it has been concluded in a previous study that sequences of opcodes are more suitable than a single opcode for representation [29]. We have considered the false negative rate (adware classified as benign) because this is more important for a user than the false alarm rate (benign classified as adware). We argue that, if a benign file is classified as adware it may not affect the system as much as if an adware application is classified as benign and thus installed on the system.

8.8.1 Algorithm Performance Analysis

The classifier generated from the IB*k* algorithm has shown the most promising results in terms of AUC and accuracy for an *n*-gram size of 4 especially for higher percentages of kept features. *k*NN and SVM are effective when the data are noisy. *k*NN has an advantage that its classification performance is refined incrementally when new training samples are introduced. J48 also has shown variance in results for smaller percentages of data. This may be attributed to the fact that for small data sets or in presence of noise, J48 is prone to over-fitting the training data.

NB has not been successful in classification as compared with other classifiers. It is evident that, as *n*-gram size and percentage of data are increased, the performance of NB classifier is varying significantly. This may be because, as the *n*-gram size increases, the number of unique combinations of opcodes in each data set increases as shown in Table 8.1. NB assigns probability to each feature. These unique combinations may be present in only a few instances and so the probability of occurrence is determined to be low in one class. However, this is not the case when using an *n*-gram size of 2 since the occurrence of any combination can be high.

For the studied problem, we may draw the conclusion that an *n*-gram size of 4 seems to be reasonable for good detection. The reason for this could be that at this size each *n*-gram is representing combination of four instructions sequences, which may be referring a function or interesting feature in the file. This is also easy to track this combination in the malware or benign files for further analysis.

8.8.2 State-of-the-Art

In a previous study on opcode-based malware detection [29], *n*=2 yielded the best results, but we argue that such short combinations of opcodes may not indicate any important function or set of instructions in the files. Due to these reasons it may be difficult to perform analysis. In another study [71], Bagging was used in conjunction with the Random Forests algorithm. However the basis for this selection was not reported. These experiments were performed on worms and viruses and have shown promising results for detection, but worms and viruses are quite different from adware in that they may exhibit clearly malicious routines; these routines can then easily be identified by human experts. But in the case of adware, the resemblance to benign software is greater. Normal characteristics of adware (such as: the displaying of ads in popup windows or the transferring of information over the network) are also present in several instances of legitimate software. Therefore, it is difficult for human experts to classify a piece of software as adware on the basis of such characteristics.

8.8.3 Opcode Analysis

We decided to use ISes rather than other common representations, such as: hexadecimal *n*-gram representations, printable strings, API calls, or messages because ISes include program control flow information. Moreover, a group of ISes may indicate an interesting function, which can be easily tracked back in the program for deeper analysis. In order to find interesting functions we analyzed models generated by SMO, JRip, NB and J48 for *n*-gram size of 4 with 70 % features. We found that most of the features that were linked to adware by other models were not considered by J48 (e.g., pushcallsbbinc and incaddandinc). This may be because J48 is considered unstable as a small variation in data set results in selection of different attributes, which affect the descendent sub trees.

8.8.4 Practical Considerations

DM techniques have performed well in detecting adware. But in the case of advanced adware with encrypted functionalities, the static analysis method used in this paper may not be successful albeit the presence of an encrypted segment can potentially be considered as an indication. It could be the case that a dynamic analysis approach has to be applied to detect these instances of advanced adware.

In terms of converting our approach into a practical solution for general users or experts, we argue that IB*k* represents a good choice of algorithm. IB*k* is the simplest algorithm with respect to working as it classifies an instance on basis of majority vote of its k nearest neighbors. The *k* is a small positive integer due to which the duration for training and building classifier from IB*k* was less than tree based and rule based algorithms where new trees or rules are required to generate or update previous. JRip algorithm was most expensive algorithm in terms of time consumed to train and generate the model, due to which this may not be considered feasible option for users. J48 algorithm was better than JRip in terms of results and time consumed for the training, but still it was expensive than other classifiers, due to which it is also not suitable candidate. SMO was nearest to IB*k*, so it may be used as an alternative or to complement the results of IB*k*. Another alternative for adware detection may be combining DM techniques with EULA analyzer as many adware vendors mention the presence of their adware in EULA to avoid legal consequence. In this way, we argue that advanced adware with encrypted routines can also be detected.

8.9 Conclusion and Future work

Many papers have been devoted to the study of detection approaches for malware such as viruses, worms, and trojans. However, less work has been done in the area of adware detection. We argue that this has little to do with the fact that adware is considered less harmful. Rather, it seems that the area of adware is avoided due to the fact that this type of software resides in a legal grey zone: some people regard adware as legitimate and others perceive adware as harmful. This paper considers the latter perception. We have presented a static file analysis method, based on operation code mining, for adware detection. A series of experiments with data sets generated using different *n*-gram sizes were performed. The experiments show promising results in terms of the area under the ROC curve (AUC) for the detection of novel instances of adware based on previously unseen examples, while maintaining a low false negative rate. The highest classification performance (AUC score of 0.949) was achieved by the k-nearest neighbor algorithm. Another conclusion inferred from these experiments is that, as the size of *n*-grams and the percentage of features are increased, the detection performance also increases. However, an *n*-gram size of 4 seems to represent a local optimum, at least for the studied algorithms. For future work, we plan to perform experiments on a larger collection of adware and benign files by introducing a hybrid identification method, which uses the combination of *n*-grams of opcodes and features extracted from EULAs. We plan to combine dynamic and static analysis techniques to be able to detect basic as well as advanced adware.

Nine

Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection

Raja Khurram Shahzad, Niklas Lavesson

Abstract

Malicious software (malware) represents a threat to the security and the privacy of computer users. Traditional signature-based and heuristic-based methods are inadequate for detecting some forms of malware. This paper presents a malware detection method based on supervised learning. The main contributions of the paper are two ensemble learning algorithms, two pre-processing techniques, and an empirical evaluation of the proposed algorithms. Sequences of operational codes are extracted as features from malware and benign files. These sequences are used to create three different data sets with different configurations. A set of learning algorithms is evaluated on the data sets. The predictions from the learning algorithms are combined by an ensemble algorithm. The predicted outcome of the ensemble algorithm is decided on the basis of voting. The experimental results show that the veto approach can accurately detect both novel and known malware instances with higher recall in comparison to majority voting, however, the precision of the veto voting is lower than the majority voting. Veto voting is further extended as trust-based veto voting. A comparison of the majority voting, the veto voting, and the trust-based veto voting is performed. The experimental results indicate the suitability of each voting scheme for detecting a particular class of software. The experimental results for the composite F1-measure indicate that the majority voting is slightly better than the trusted veto voting while the trusted veto is significantly better than the veto classifier.

9.1 Introduction

Malicious software (malware) is a common computer threat and is usually addressed through the static and the dynamic detection techniques. Anti-malware tools are only able to detect known malware instances and the success rate is circa 30 % [11] in the wild. In an effort, to extend both the static and dynamic approaches, some researchers apply machine learning (ML) algorithms to generate classifiers, which show promising results both in detecting the known and novel malware. To increase the detection accuracy, the output (prediction) of different classifiers is combined (based on different parameters) to form an ensemble [114]. Ensembles can be data dependent such as multiple algorithm trained on the same data set, or independent from the data, i.e., using statistical measures [115]. The prediction of each participating classifier in the ensemble may be considered as a vote for a particular class, i.e., benign class or malware class. The ensemble's outcome is generally derived on the basis of different voting strategies. Different voting strategies may give different results depending upon different factors such as families of algorithms used. Among different voting strategies, the majority voting is generally used for different problems. The majority voting is considered as a simplest and effective scheme [116]. The majority voting scheme follows democratic rules, i.e., the class with highest number of votes is the outcome. Majority vote does not assume prior knowledge about the problem in hand or classifiers used and may not require training [116]. The majority voting scheme has different limitations such as a subset of classifiers (majority in number) may agree on the misclassification of an instance by a chance. An alternative voting scheme is the veto voting, i.e., one single classifier vetoes the decision of other classifiers. The veto voting scheme is used in the fault diagnosis, the author identification and the malware detection [117, 118]. For the malware detection a ML-based detection model is proposed in which the inductive biases of different algorithms are combined, and the final prediction is given on the basis of veto voting, i.e., if an algorithm predicts the instance as a malware, this prediction may veto all the other predictions and the outcome is malware [118]. The veto voting may also affects the performance of the ensemble as outcome may depend on one single algorithm. However, the veto voting can achieve higher classification accuracy on the assumption that the data set contains abundant instances of a particular class (favored by veto) [119]. Thus, it is worth to investigate which voting scheme, i.e., majority voting or veto voting is suitable for the malware detection. After the comparison of the majority voting and the veto voting, the veto voting is extended from a simple veto voting to the trust-based veto voting. The trust-based veto voting considers the trust of each algorithm to determine whether an algorithm or set of algorithms can veto the decision. The trust-based veto voting is also applied on the same data set and results are compared with the majority voting and the veto voting. The majority voting is suitable for detecting benign applications and lacks the accuracy in the detection of malware in comparison to the veto voting. Similarly, the veto voting is suitable for detecting the malware and is less accurate for the detection of benign. The experimental results indicate that the proposed trust-based veto voting algorithm may be used to overcome the deficiencies of both the majority voting and the veto voting up to some extent. The trust-based veto voting accurately detect both novel and known instances of malware better than the majority voting and accurately predicts about benign instances better than the veto voting. The experimental results also indicate that if composite measure is taken into account, majority voting is slightly better than the trust-based veto voting.

9.1.1 Aim and Scope

The aim is to evaluate the malware detection methods that combine the output of a set of classifiers and provides a classification on the basis of the majority voting or the veto voting. A malware detection application is developed, which implements the majority voting, the veto voting and the trust-based veto voting. The prediction results from all voting strategies are compared with each other. To achieve better predictive results, the quality of information (i.e., the information, which provides valuable input for classification) derived from the data in the pre-processing stage is very important. Therefore, two pre-processing techniques are also proposed and investigated.

9.1.2 Contribution

The contributions are: first, a malware detection model is proposed and implemented, which combines the inductive biases of different algorithms and uses contrary voting strategies for the prediction. Second, an extension to a particular voting strategy, i.e., the veto voting is proposed and implemented. Third, the empirical results of different voting strategies are analyzed. Fourth, two pre-processing techniques are proposed to extract features from executable files. These pre-processing techniques can be used to extract both hexadecimal based features or assembly instruction based features of different sizes.

9.1.3 Outline

The remainder of the article is organized as follows: Section 9.2 discusses the background, terminology and related work. Section 9.3 presents the veto-based classification by discussing it's architecture. Section 9.4 discusses the pre-processing techniques. Section 9.5 describes the experimental procedure for the first experiment (i.e., Experiment I) and compares the results of the majority voting and the veto voting. Section 9.6 describes the trust-based veto voting, presents the second experiment (i.e., Experiment II), and analyzes the experimental results. Finally, Section 9.7 concludes the work and describes future directions.

9.2 Background

One of the challenges faced by computer users is to keep the information and communication away from unwanted parties who exploit vulnerabilities present in the operating system (OS) or third party software to jeopardize the communication and access the information. A popular way to exploit vulnerabilities remotely is by using a malware [1]. Traditionally, the malware detection is conducted either by using the static analysis, i.e., by matching specific patterns called signatures or on the basis of a rule set, or the dynamic analysis, i.e., changes occurring in the system due to the execution of a specific file. The main deficiency of these techniques is that they fail to detect the zero-day attacks.

The use of ML has been investigated in fields such as natural language processing, medical diagnosis, and malware detection. ML can be divided into two broad categories. In supervised learning, algorithms are used to generate a classifier or a regression function using the labeled data. To achieve the better predictive performance, a finite set of classifiers can be combined as an ensemble. The prediction of most ensembles is based on the majority voting [114]. If the data is incompletely labeled, unsupervised learning is used. To achieve better predictive performance in unsupervised learning, deep learning can be used. In deep learning, algorithms learn from different levels of representations to find the complex patterns in the data.

Any algorithm used in supervised or unsupervised learning has its own inductive bias. Inductive bias of learning algorithms refers to the set of assumptions that a learning algorithm uses for predicting the output of unseen inputs [39]. In other words, it is a set of assumptions that is used by a learning algorithm to prefer one hypothesis over the other hypothesis in the search space, in order to find a suitable hypothesis which can provide better predictions for the problem in question. These factors affect the classifier performance [38]. In the case of malware classification, an algorithm may be more accurate in classifying viruses than adware. Due to these reasons, detection results may vary from one learning algorithm to another learning algorithm. Therefore, it may be appropriate to join the results of classifiers trained at different representations to achieve the improved accuracy in predicting the class of unseen instances.

9.2.1 Terminology

Data set

A piece of information from a particular object such as a binary file or an image in a specific format is called a feature. The format of the feature is called feature representation. Each instance present in the original data is represented by its specific features for the ML processable data sets. For the malware detection task, different features can be used to represent the binary files such as files may be converted into the hexadecimal code [20,33] or assembly instructions [24] to produce the data sets. Features may also be extracted from the binary files such as printable strings or system calls [33] to generate the data set.

Feature Selection

To improve the accuracy of ML algorithms, complexity of the data sets is reduced. For this purpose, the most common approach is to apply a feature selection algorithm, which measures the quality of each feature and prioritize features accordingly [38]. Only features that can provide the valuable information for the classification are kept, and the rest are discarded. In the malware detection, an extensive feature set is produced from the binary files data set. This feature set contains many invaluable features, which can degrade the performance of a ML algorithm. Therefore, it is necessary to obtain a subset of valuable features by applying the feature selection.

Classification

In ML, the classification is divided into two stages, i.e., training and testing. Learning algorithms are applied on the training set to build a model (commonly called classifier) [39]. This stage is called training. During the testing stage, the generated classifier is used to predict the class of unseen instances.

Ensemble

Ensemble are capable of combining multiple models for the improved accuracy [38]. The different models for the ensemble may be generated from the same base algorithm on different subsets of the data or different algorithms on the same data set. Ensembles perform better than a single model due to the diversity of base models [120].

Trust

Trust is primarily related to the human behavior of believing a person to meet expectations. Trust has different meanings in different contexts. Trust can be defined as *"Trust is quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context"* [121]. For the problem in hand, the terms trustor and the trustee refers to an algorithm that quantify the trust. Trust can be quantified as +1 or -1; the increased or decreased value can assist in determining the extent of the trust. The trust can be quantified as the single trustor or the group trust. The trust especially the group trust is quantified for different computational problems such as for the authentication, in the peer-to-peer networks, in the mobile ad-hoc networks, for resisting the different network attacks and for spam emails.

9.2.2 Related Work

A significant amount of research for classification tasks has applied techniques ranging from statistical methods to machine learning like supervised learning and deep learning (DL). The use of DL has been investigated to learn different levels of the representation in order to model complex relationships in the data to classify patterns and objects [122]. DL has also been used to extract features and represent them at different layers with different representations or abstraction to be used for vision, face recognition, and hand written digit recognition. Similarly, the layered architecture has also been used for detecting the malicious behavior [11]. In some cases, the decision from an individual model or even from several models may be insufficient to obtain the final predication, especially when the cost of misclassifying one class is higher than misclassifying the other class. As a solution, a few researchers have used veto voting for automated identification of a disease pulmonary embolism [123] and authorship identification [124].

For the malware detection, several researchers have used ensemble based on the majority voting. The majority voting is compared with different ensemble methods on five different malware data sets [125]. The majority voting is also used to generate the rules, according to the Dempster-Shafer theory, to classify the malicious codes based on the n-gram features of the binary files [126]. Some researchers have applied the variation of majority voting such as the weighted majority voting for the malware detection [67]. The concept of the veto voting is not investigated for the malware detection. Therefore, it is worth to investigate the veto voting for the malware detection.

To support the veto voting, the concept of trust may be used. In an early work on the authentication in open networks, the trust is used to accept or reject an entity for a task [127]. A set of inference rules is used to determine the value of trust, i.e., $0 \leq trust \leq 1$ and derived value is further used for the decision. Both direct and group trusts are used. In a study, trust value is used for resisting the non-trivial attacks on the authentication of the origin of the message in the distributed system [128]. A quantitative framework based on the attack values for resisting the attack is proposed [129, 130]. The proposed framework uses the group trust metric and calculates a trust value for all the nodes simultaneously. The values are further used to build a distributed name server, verify the meta-data in peer-to-peer networks, and resistance to the Spam e-mails. The authors also present a real world example, i.e., Advogato website[1].

[1]http://www.advogato.org/

A common trust based algorithm for the peer-to-peer network is the EigenTrust algorithm [131]. The EigenTrust algorithm uses peer nodes to assign the trust to each node. The assigned trust is used to computes global trust values in a distributed manner and node-symmetric manner. Global trust value is also used to distinguish malicious nodes in the system. The priority is given to the opinion of high reputation nodes. The EigenTrust algorithm is used to proposed a non-manipulable trust system for peer-to-peer networks [132]. The authors propose a partitioning technique that is used to partition the peers in groups and incentives for the peers to share files. Mobile ad hoc networks are decentralized networks and nodes in such network cooperate with each other [133]. The trust value of each node is used to improve the security of network. The authors use the local trust and recommendation trust, which are combined to obtain the combination trust. Finally, the combination trust value is used to evaluate the level of risk for ongoing tasks. In the field of multi-agent systems and ML, trust is used to make a reputation system for auction systems [134]. A generic ML based trust framework is proposed to calculate the trust of a transaction by an agent. The trust is calculated on the basis of previous successful transactions based on distinguishing features of successful and unsuccessful transactions. The distinguishing features are given as input into ML algorithms to extract the relationship. The extracted relationships are further used to predict about the success of the current transactions.

9.3 Veto-based Classification

In certain situations, the recommendation from more than one expert may be required. In such cases, a committee of experts is formed as it is expected that a committee always performs better than a single expert. Normally the committee uses the majority voting for combining the decisions of experts to reach a final conclusion. In some cases, the committee may grant the right to veto the decision of the committee to any member. In ML, multiple algorithms can be used to generate multiple classifiers (experts) for a classification task. Every classifier has its own inductive bias, which affects the predictive performance. Research results indicate that ensemble perform better than single classifier in fields of text categorization [135] and data classification, etc. Several rules such as majority voting, i.e., bagging [38], weighted combination where weights represent effectiveness of member classifiers such as boosting [38], dynamic classifier selection [136, 137] and the veto voting [123, 124] can be used for combining the decisions and having a final prediction. Veto voting is used to give importance to a single expert (classifier) who predicts against the majority.

In malware detection, ignoring the correct prediction about a malicious file from a single expert may incur a higher cost in terms of security violations. The security violations may cause serious data loss, privacy or monetary damages to the user. Therefore, a veto voting based classification model is more appropriate than a majority voting based model. A model is proposed, which combines the inductive biases of individual classifiers and the final decision is given on the basis of veto voting. For brevity, the veto voting based classification system is referred to as veto classifier in later sections.

9.3.1 Voting Rules

The main objective of veto based classification is to combine the multiple predictions to reach a final decision. Formally, a veto based classification system consists of candidate classifiers (C) set and vote set V. The set of candidate classifiers C, and the set of votes (V) is a finite set (C, V) with predetermined fixed number of maximum classifiers and votes. The vote from each classifier is considered on the basis of rules given below. Some rules mentioned below are also recommended for the general voting [138].

Anonymity All votes in the vote set (V, a finite set with a predetermined number of votes) are treated equally, and the outcome of the classifier committee remains consistent with any permutation of the votes.

Neutrality All candidates in the classifier set (C, a finite set with a predetermined number of classifiers) are considered equally without any additional weighting.

Independence Candidate classifiers are independent of each other, and the outcome of the voting system remains consistent with any combination of classifiers with votes, i.e., $(C_1, V) \cup (C_2, V) \subseteq (C, V)$ where C_1 and C_2 are different combinations of classifiers.

Completeness All votes from the classifiers are considered and counted only once.

Veto Any vote indicating an instance as malware, alone can determine the outcome of the classification task regardless of the number of other votes.

Consistency The result of the voting remains consistent even if the base classifiers are split into two disjoint sets and each set vote separately. The votes from each subset create a single vote set. Formally this can be mentioned as $(C, V_1) \cap (C, V_2) \subseteq (C, V)$ where V_1 and V_2 are the partitions of votes.

Usability Voting procedure can be used for other similar problems.

Verifiability Any user of the voting system can verify the outcome of voting by counting the votes manually.

Ideally veto based voting performs better than the single classifier because if any classifier in the committee predicts the class of an instance as malware, it may veto all the other predictions from the other classifiers. For the neutrality, the results from all the classifiers are combined without any additional weighting or filtering. It is also possible to use weighting method [139] for the votes such as by assigning more weight to the vote of a classifier who outperformed all other classifiers in terms of accuracy during the training stage.

9.3.2 Architecture

The model can be implemented in two possible ways, i.e., N-layers implementation and parallel implementation. N-layers implementation is based on the serial implementation.

N-Layers implementation The model can be implemented in n-layers with any permutation of classifiers, see Figure 9.1. Each layer can be customized with different n-gram sizes, several feature representations, various feature selection algorithms and learning algorithms. It is recommended that different classifiers shall be used while maintaining their neutrality as much as possible to increase the effectiveness [120]. From the lower layer, the instances that are declared as benign are given to the upper layer for the reclassification. In each layer, all classifiers give their predictions about the instances from the lower layer (or data sources). If at any layer, an instance is classified as malware, it is not fed into the next layer. The classification results from all the layers are given to the veto classifier. The malware prediction at any layer for any instance may be considered as veto for that particular instance. However, the final decision about the class of particular instance is taken by the veto classifier.

Parallel Implementation Instead of using layers, all the possible permutations of classifiers can be implemented in a distributed or parallel manner. Each learning algorithm is trained over the entire data set for generating the classifiers. Instead of testing only positive instances by some classifiers, each classifier works independent of results from other classifiers. All the votes from the classifiers are collected at a central location where the veto classifier outputs the final prediction.

9.4 Pre-processing Techniques

The selection of representative features affects the predictive performance of a classifier. Consequently, two n-gram extraction techniques

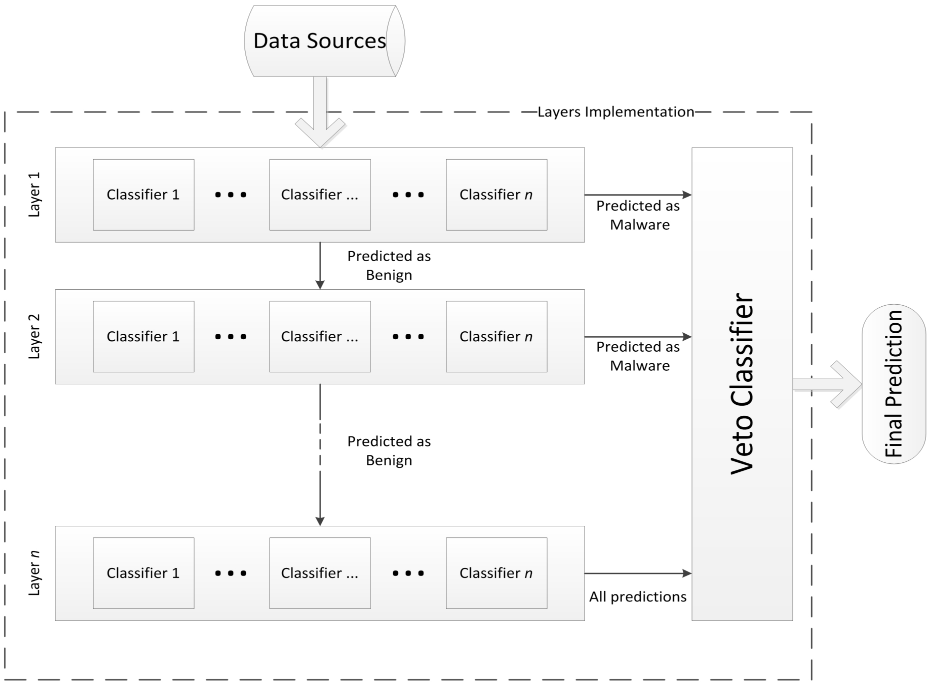


Figure 9.1: *N*-layer Implementation

are proposed. Most of the research studies for the malware detection demonstrate the use of either the hexadecimal-based *n*-gram data set or the opcode-based *n*-gram data set for an experiment. The proposed extraction techniques can be used to extract both representations. For this study the opcode *n*-gram extraction is performed, therefore, the proposed techniques are explained in the context of opcode *n*-grams only.

Traditionally the *n*-gram extraction is performed by using a fixed size window with a fixed step; the step size is equal to the window size. The fixed size window traverse the data file to extract the specific size *n*-grams. The generated output contains adjacent *n*-grams. To explain this process, assume that a disassembled binary file contains the following given data. A pair of characters represents an opcode (or a hexadecimal code). The task is to create *bi*-grams, i.e., *n*-gram of size 2 from this data file.

aa bb cc dd ee ff gg hh ii jj kk ll mm nn oo pp

The generated *bi*-grams from this file are "*aabb ccdd eeff gghh iijj kkll mmnn oopp*" and so on. The fixed size window is unable to extract some *n*-grams such as "*bbcc*" or "*ddee*". If the file size is large and the data is redundant then there is a probability to have missing combinations, but still missing *n*-grams cannot be produced in the appropriate frequency and can have less importance for the classification task.

9.4.1 Overlapping *n*-grams

To address the problems of missing *n*-grams, the use of configurable sliding window to generate overlapping *n*-grams is proposed. The window can be configured with two parameters, i.e., size and step. The size parameter defines the size of a *n*-gram to be extracted, and the step parameter defines the number of opcodes to be skipped before extracting the next *n*-gram. It is expected that all possible combinations can be extracted by this extraction technique. Referring the example in Section 9.4, if the window is configured as following, size = 2, i.e., two adjacent opcodes are extracted to form a *n*-gram and step = 1, i.e., after the *n*-gram extraction window skips one opcode (first opcode) to move forward. This configuration generates "aabb *bbcc* ccdd *ddee* eeff *ffgg* gghh *hhii* iijj *jjkk*" and so on.

9.4.2 Non-adjacent Opcode Extraction

Either by using the traditional *n*-gram extraction or the overlapping *n*-gram extraction, extracted *n*-grams can provide the information only about the dependencies of the adjacent opcodes. It is valuable to look at the information provided by non-adjacent opcodes. Non-adjacent opcodes have dependencies such as they can be function header and tail. Some changes are proposed in the overlapping *n*-gram extraction method to explore the information both from non-adjacent opcode and non-adjacent adjacent opcode. The size parameter is changed to the start-end size parameter. The start-end size parameter defines the number of adjacent opcodes to be extracted for the start and the end of a *n*-gram. The step size parameter defines the number of opcodes to be skipped for extracting a new *n*-gram. A new parameter is introduced, i.e., gap size, which specifies the interval between start and end opcode or number of opcodes to be skipped between the start and the end opcode of a *n*-gram. The example mentioned in the Section 9.4 can be used to describe this procedure. If the window is configured as following for extracting non-adjacent *bi*-grams, start-end size = 1, i.e., one opcode for the start and one opcode for the end of a *n*-gram are extracted, step = 1 and gap = 1, i.e., one opcode between the start opcode and the end opcode of a *n*-gram is skipped. This configuration produces the *bi*-grams, which contains non-adjacent opcodes. The generated output is "*aacc bbdd ccee ddff eegg*" and so on. To have non-adjacent adjacent opcodes in a *n*-gram, the configuration can be changed as follow: start-end = 2, i.e., two adjacent opcodes for the start and the end of a *n*-gram are extracted; the step size and the gap size are kept 1. The generated output is "*aabbddee bbcceeff ccddgghh*" and so on. If the value of the gap size and the step size parameters is changed from 1 to 2, the generated output is "*aabbeeff ccddgghh eeffiijj*" and so on.

9.5 Experiment I

The aim of the experiment is to evaluate the proposed veto voting based malware detection method and impact of the proposed data pre-processing techniques and compare the results with the majority voting. The proposed method can be used to detect either a specific type of malware or different types of malware; however in this study a single family of malware is used. The experimental data set contains Windows-based executable files. Windows is a common OS for novice users and contains

Table 9.1: Experiment with one data set and three algorithms

Data Set[a]	Algorithm	TP	TN	FP	FN	R[b]	P[b]	F1[b]
***n*-gram**	JRip	243	184	66	07	0.972	0.786	0.869
	J48	226	225	25	24	0.904	0.900	0.902
	IB*k*	224	225	25	24	0.896	0.899	0.897
	Veto[c]	243	203	47	07	0.972	0.837	0.900
	Majority[c]	223	233	17	26	0.895	0.929	0.912
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
	J48	232	234	16	18	0.928	0.935	0.931
	IB*k*	224	224	26	26	0.896	0.896	0.896
	Veto	246	208	42	04	0.984	0.854	0.914
	Majority	230	240	10	20	0.920	0.958	0.938
S. Window	JRip	209	215	35	41	0.836	0.856	0.846
	J48	215	205	45	35	0.860	0.826	0.843
	IB*k*	164	237	13	86	0.656	0.926	0.768
	Veto	242	139	111	08	0.968	0.685	0.802
	Majority	220	204	46	30	0.880	0.827	0.852

[a] The full names of data sets are *n*-gram data set, overlap data set and sliding window data set.

[b] R is Recall, P is Precision, and F1 is F-Measure.

[c] Veto is Veto Classifier and Majority is Majority voting.

[d] Veto Classifier and Majority voting both are applied on all the three data sets.

different vulnerabilities[2], which can be exploited by a malware. When a binary file in the data set is disassembled, different file features such as assembly language instructions and printable strings, are produced in the text format, which are further processed to extract the assembly directives, i.e., opcode. Opcodes are further processed to produce the *bi*-gram data sets using different strategies. Different text categorization techniques can be applied to the output generated in the previous step to get discriminating features of benign and malware. Term Frequency-Inverse Document Frequency (*tf-idf*) is used to derive the significant features from the data sets. The extracted features are used to create

[2] http://technet.microsoft.com/en-us/security/bulletin/

Attribute-Relation File Format (ARFF)[3] files. ARFF file is a structured ASCII text file that includes a set of data instances, each described by a set of features [38]. ARFF files are used as input to the proposed model, which uses Waikato Environment for Knowledge Analysis (Weka) application programming interface (API) [97] for applying learning algorithms to build and analyze classifiers. A pre-experiment is performed for the selection of learning algorithms. The first experiment is divided into two sub-experiments. In the first experiment, the inductive biases of the different classifiers built on the same data set are combined. Second experiment combines the inductive biases of individual classifiers built on different data sets. In both experiments, the results from all the classifiers are given to the veto classifier for the final prediction.

9.5.1 Feature Representation

Opcode is used for generating *bi*-grams as features. It is concluded in the previous studies that opcode *n*-grams are better choice for the malware detection in comparison to other features such as printable strings, systems calls or byte code (hexadecimal) *n*-grams [24]. The opcode *n*-grams are capable of providing the information about the program flow, structure and function that cannot be deduced from other representations.

9.5.2 Data Set Creation

For the experiment, scareware (rouge) software is selected as malware representation. The reason for this choice is, there is a subtle difference between scareware and benign. In case of traditional malware, presence of malicious payload distinguishes a malware from the benign. However, in scareware no specific malicious payload is available that can be used to differentiate a scareware from the benign. Absence of malicious payload may deceive human expert for the classification of a particular software as scareware.

[3]http://www.cs.waikato.ac.nz/ml/weka/arff.html

Scareware are scam software that usually masquerade as an anti-virus software and resembles the benign software in functionality. Scareware generates the false alarm about the presence of malware in the user's machine. The false alarms are used to scare the users into disclosing their credit card information for buying the protection[4]. No public data set e.g., virus, Trojan, and worm data sets is available for the scareware detection experiments. Therefore, a data set with 500 files is created; out of which 250 files are scareware, and 250 files are representing benign. The benign files are default applications of Windows OS such as notepad, paint and applications available online for download at CNET Download[5]. All the benign files are scanned with commercial security software (anti-malware) to reduce the chances of malware presence in a benign file. Scareware files are obtained from the malware database of Lavasoft[6].

9.5.3 Pre-Processing

The disassembled file is a standard text file, which contains three fields, i.e., the memory address of the instruction, the byte-based location in the file and the instruction itself (combination of opcode and operands). The next step is to extract only opcodes from the files and discard irrelevant information, i.e., operands. The extracted opcodes are saved in the original order. After opcodes extraction from the disassembled files, three different procedures are used to tokenize the data to produce *bi*-grams for three different data sets. Each row in a data set represents a *bi*-gram, i.e., concatenation of two opcodes. Hereafter, these three data sets are referred to as *bi*-gram data set, overlap data set and sliding window data set respectively to indicate the method used in creating that particular data set. The *bi*-gram size has yielded the best performance in a previous study [29] and possible combinations of opcodes to produce *bi*-grams are limited, depending upon the number of reserve words in the assembly language.

[4]http://news.bbc.co.uk/2/hi/8313678.stm
[5]http://download.com
[6]http://lavasoft.com

For the *bi*-gram data set, a fixed size window traverse each input file from top to bottom. In every step, a *n*-gram consisting of two opcodes is extracted and recorded in another file having the similar file name, but different extension. The purpose of keeping the similar name is to keep track of benign files and scareware files, so each file can be represented at the same position in all three data sets and finally in the ARFF file. For overlap data set method mentioned in the Section 9.4.1 is followed with the configuration, i.e., size = 1 and step = 1. For the sliding window data set, start-end size and step parameters are kept one. To obtain the nonadjacent opcode *bi*-grams, each file is processed in four consecutive passes with a gap size ranging from 1-4. Due to the changing gap size, the first generated *bi*-grams are having a gap of one opcode between the start opcode and the end opcode, in the second pass there is a gap of two opcodes and so on. The process of generating the sliding window data set is slower than generating the *bi*-gram data set and the overlap data set. However, the computational cost and memory requirements for generating the sliding window data set are lower than creating large size *n*-grams.

9.5.4 Feature Selection

Many real world problems are complex. To apply learning algorithms, the dimensionality of the complex problem is reduced by choosing a subset of significant features from the given set of (raw) features. The selected subset of features plays significant role in the increase/decrease of either classification and/or computational performance. Significant feature selection is done by using a feature selection algorithm, removing features that are deemed unlikely to improve the classification process. In the field of text classification, *tf-idf* shows promising results for the valuable features selection. In this experiment *tf-idf* is applied on data sets to limit the number of features to top 1000 features per data set. The *tf-idf* is a statistical measure of importance of a *bi*-gram in the entire data set [140]. The *tf* is the number of times a *bi*-gram occurs in a file; *df* is the number of files in a class that contain a specific *bi*-gram.

The *idf* of a *bi*-gram is obtained by dividing the total number of files (N) by the *df* and then taking the logarithm.

9.5.5 Performance Evaluation Criteria

Each learning algorithm is evaluated by performing cross-validation tests. Confusion matrices are generated by using the responses from the classifiers. The following four estimates define the elements of a confusion matrix: True Positive (TP) represents the correctly identified scareware programs. False Positive (FP) represents the incorrectly classified benign programs. True Negative (TN) represents the correctly identified benign programs, and False Negative (FN) represents the incorrectly identified scareware programs. The performance of each classifier is evaluated using Recall (R), which is the ratio of scareware programs correctly predicted from the total number of scareware programs, Precision (P), ratio of scareware programs correctly identified from the total number of programs identified as scareware. F-Measure (F1) is the harmonic mean of the precision and the recall and is the final evaluation measure.

9.5.6 Pre-Experiment for Algorithm Selection

A number of studies have addressed the similar problem with different learning algorithms; however, none of the authors is conclusive on the choice of algorithms either for the malware detection or according to the produced data set. In a number of studies Ripper (JRip) [91], C4.5 Decision Tree (J48) [106], *k*-nearest neighbor (IB*k*) [109] Naive Bayes [38] and SMO [107] outperformed other algorithms. Based on previous research, a pre-experiment is performed to evaluate all these algorithms on all the three data sets. The top three algorithms, i.e., JRip, J48 and IB*k* are considered as candidates, to combine their inductive biases for the final prediction in the proposed model.

9.5.7 Results and Discussion

In the first experiment, one data set is used to build classifiers from three different algorithms. In the second experiment, three data representations are used and one classifier is trained from each representation. Majority voting is compared to the veto voting. In the first experiment, the predictions from three classifiers are collected and given to the veto classifier and the majority voting. The predictions from all the classifiers including both voting strategies on each data set are shown in Table 9.1. In the second experiment, three algorithms, i.e., JRip for *n*-gram data set, JRip for the overlap data set, and J48 for the sliding window data set are selected on the basis of the recall in the first experiment. These algorithms are used to built three classifiers and the predictions about each instance from these classifiers is given to the veto classifier and the majority voting for the final prediction. The results of this experiment (see Table 9.2) indicate that the recall of the veto classifier is better than the recall values in the first experiment. Majority voting shows the similar behavior for the precision.

The experimental results indicate that combining the inductive biases of different algorithms trained on multiple representations predicts better for the malware detection than combining the inductive biases of different algorithms trained on the same data set. The experimental results of both voting strategies can be discussed in three dimensions by using three measures, i.e., recall, precision, and f-measure. The experimental results show that the veto classifier has better recall than the majority voting, i.e., veto classifier reduces the number of misclassified scareware. Recall is the key measure as the objective of the veto approach is to reduce the likelihood of malware misclassification while tolerating a percentage of false positives or decrease in the precision. If the system is tuned to predict all applications as malware, it will produce a high false positive rate, which is undesirable from a user's point of view. Users need the accurate prediction both for the malware and benign applications. Therefore, the precision is also considered as a complimentary measure with the recall. The veto classifier shows a higher tendency for the correct detection of scareware while the majority voting shows a tendency towards the detection of benign applications. Therefore, the precision rate is higher for the majority voting. There are few instances, which are misclassified by both voting schemes. Most of these instances are benign, but predicted as scareware by both the veto classifier and the majority voting. However, the number of such instances is minimal. The precision and the recall have an inverse relationship if the precision increases, the recall decreases. Therefore, another evaluation measure is required, which combines the precision and the recall. Thus, the final evaluation measure is F-measure, which evenly weights the precision and the recall. It may be argued that the arithmetic mean of the precision and the recall can also be used as a composite measure. However, the arithmetic mean is an inappropriate measure as with 100 % R and 0 % P or vice versa; the arithmetic mean is always 50 %. This is not the case with the harmonic mean as the harmonic mean is always less than or equal to the arithmetic mean [141]. If there is a difference between the value of R and the P such that the value of R is significantly smaller than P, the harmonic mean tends strongly towards the recall. F1 of the majority voting is higher than the veto classifier, which favors the use of majority voting for the problem in question. However, the recall for the majority voting, which is a key measure, is lower than the veto classifier so it may be argued that the veto is a better approach for the malware detection. Thus, the veto classifier shall be extended to increase the precision.

Bi-grams are used as the feature in the experiment because they are computationally inexpensive to produce. Generally such short combinations may not represent an important function or set of instructions in the files and are difficult to analyze. However, *bi*-grams in the sliding window data set can provide the valuable information for the scareware analysis due to the combination of non-adjacent opcodes. Scareware resembles the benign applications such as displaying popup windows or alert messages, and showing the dialog boxes. Therefore, it is difficult for the human experts to predict about the scareware by analyzing the functionality of an application only. The proposed model helps the hu-

Table 9.2: Experiment with three data sets and one algorithm on each data set

Data Set[a]	**Algorithm**	**TP**	**TN**	**FP**	**FN**	**R**[b]	**P**[b]	**F1**[b]
n-gram	JRip	243	184	66	07	0.972	0.786	0.869
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
S. Window	J48	215	205	45	35	0.860	0.826	0.843
	Veto[d]	248	195	55	02	0.992	0.818	0.896
	Majority[d]	223	247	03	26	0.895	0.986	0.938
	Trust Veto[e]	235	230	20	15	0.940	0.922	0.931

[a] The full names of data sets are *n*-gram data set, overlap data set and sliding window data set.

[b] R is Recall, P is Precision, and F1 is F-Measure.

[c] Veto is Veto Classifier and Majority is Majority voting.

[d] Veto Classifier,Majority voting and Trust-based veto Classifier are applied on all the three data sets.

[e] Trust Veto is Trust-based Veto Classifier.

man expert by automating the process of analyzing and predicting the scareware (malware). JRip and J48 algorithm are considered expensive algorithms in terms of time consumed to train and generate the model. However, it is easy to analyze the rules and trees generated to differentiate the scareware and benign.

The decisions of the different classifiers are combined to produce better results, and such combination shall not be considered as a substitute of a good classifier [142]. The proposed veto classifier follows the same principle. Veto classifier is neither a substitute of a good classifier nor replacing the majority voting. In the domain of decision theory, it has been suggested that different voting strategies shall be adopted for different tasks according to the problem in question. We argue that the veto classifier is a better choice for the malware detection task as this approach addresses the problems of the majority voting. There are different problems related with the majority voting such as majority voting may ignore the right decision of the minority. While ignoring the decision from the minority votes, the total number of majority votes may have an ignorable difference in comparison with the total number of minority votes. Another problem of the majority voting is the choice of the number of candidate classifiers. If the number of selected classifiers is an odd, then a simple majority can be obtained, but if the number of selected classifiers is an even then a situation may arise where equal numbers of votes are given to both the benign and malware classes. In the domain of ML, different variations of majority voting has been suggested such as restricted majority voting, enhanced majority voting, and ranked majority voting to address the problems of majority voting [143]; such problems are avoided with the proposed veto classifier.

The results of the veto classifier depend upon a suitable permutation of the algorithms. Some permutations may obtain 100 % recall by just predicting all applications as malware. Some permutation can achieve 100 % precision, if all the instances are predicted as benign applications. Before permutation, classifiers selection is a critical and complex task. For a small number of classifiers, an optimal combination can be found exhaustively, but as the number of classifiers increases, the complexity of selection is increased due to their different inductive biases, search limits and practical applicability. The classifier selection process can be improved by a static selection or dynamic selection method [144].

9.6 Experiment II

Results of the experiment I in the Section 9.5.7 suggest that the veto classifier is a better choice for the malware detection problem. Results also indicate that the majority voting skews towards benign applications and the veto classifier skews towards malicious applications. In the Experiment I, all algorithms are treated equal for the final prediction to assure the "neutrality" property for the veto classifier. Consequently, algorithms who generally demonstrate higher misclassification in comparison to other algorithms (generally referred to as weak learners), supersede the correct prediction. This phenomenon produces a high false positive rate and the precision of the veto classifier is significantly lower than the majority voting. To address the above mentioned problems in the veto classifiers, use of the algorithm's "trust" is suggested and the veto classifier is extended as a trust-based veto classifier. The aim of this experiment is to evaluate the trust-based veto classifier for the malware detection and compare the performance of the proposed algorithm with the majority voting and the veto classifier.

9.6.1 Trust-based Veto algorithm

Trust as a quantitative measure can be quantified with integer values. The positive integer may represent the trust while a negative integer may be used to represent the distrust. The quantified trust value in computational problems can be calculated for participating nodes, algorithms, and agents. For the machine learning problems, different kinds of trust can be calculated for algorithms contending in the system. Consequently, an algorithm is proposed, which generate trust-based veto classifier for the malware detection. The trust-based veto algorithm involves three kinds of trust, i.e., local trust (can also be referred to as direct trust), recommended trust, and global trust. Each algorithm in the system calculates its trust level for other algorithms in the system, i.e., how much $algorithm_x$ trusts the $algorithm_y$ in terms of predicting the class of an instance, called local trust (t), see Figure 9.2. The local trust value is further used to calculate the recommended trust (RT) for each algorithm, see Figure 9.2. The recommended trust aids in calculating the global trust value (GT) of each algorithm. The global trust value is used for having a veto decision.

Local Trust Calculation

Local trust of an $algorithm_x$ on $algorithm_y$ ($t_y : algo_x \to algo_y$) is calculated by comparing the predictions (d) of both algorithms with each other and the actual class (C) of the instance, see Algorithm 1. Suppose, from a data set of benign and malicious instances, an instance of benign class is given to the $algorithm_x$ and the $algorithm_y$ for predicting the class of the instance. There is a finite set of possible predictions, i.e., both algorithms may predict correct, or both algorithms may predict

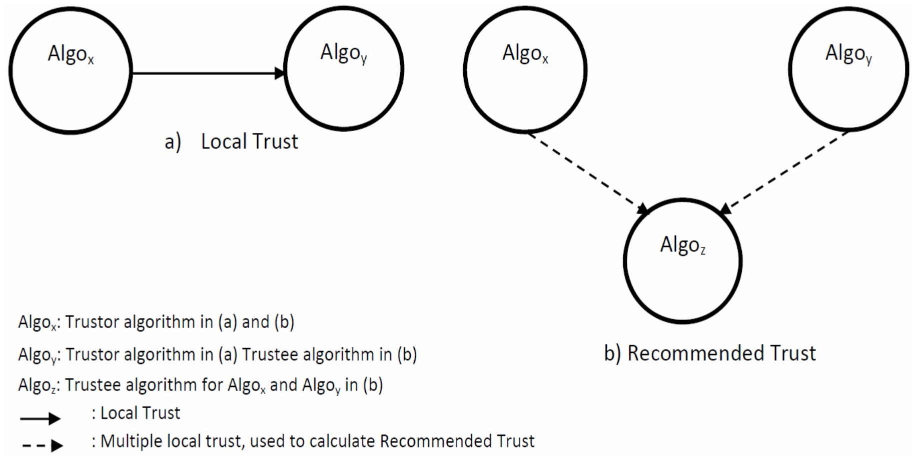

Algo$_x$
Algo$_y$
a) Local Trust
Algo$_x$
Algo$_y$
Algo$_z$
b) Recommended Trust
Algo$_x$: Trustor algorithm in (a) and (b)
Algo$_y$: Trustor algorithm in (a) Trustee algorithm in (b)
Algo$_z$: Trustee algorithm for Algo$_x$ and Algo$_y$ in (b)
: Local Trust
: Multiple local trust, used to calculate Recommended Trust

Figure 9.2: The direct trust and the recommended trust of algorithms

incorrect, or any one of the algorithms may predict the correct class. If both algorithms have the same prediction for the instance, either correct or incorrect; trust is not affected. However, if the $algorithm_x$ predicts the incorrect class and $algorithm_y$ predicts the correct class, the $algorithm_x$ increases the trust level (*sat*) of the $algorithm_y$ with +1. In case, the $algorithm_x$ predicts the correct class and the $algorithm_y$ predicts incorrect class, the $algorithm_x$ increase the distrust level (*unsat*) of the $algorithm_y$ with +1. All the instances in the data set are given to both algorithms sequentially for the prediction. At the end of process, local trust of the $algorithm_y$ is calculated by dividing trust (*sat*) with the sum of the trust (*sat*) and the distrust (*unsat*), see Algorithm 1.

Recommended Trust Calculation

The local trust shows the unique trust on a particular algorithm (e.g., $algo_y$) from another algorithm (e.g., $algo_x$). This value varies from the algorithm to the algorithm in the system and cannot be used as a final metric for deciding about a veto in the system. The Recommended trust

Algorithm 1 Trust Calculation

```
Require: Actual Class of Instance ($C$), prediction of $algo_x$ ($d_x$), prediction of $algo_y$ ($d_y$)

function LOCALTRUST
    repeat
        if $dx = dy$ then    ▷ Prediction of both algorithms may be correct or incorrect
            $movenext$
        end if
        if $dx \neq dy$ then    ▷ Compare the prediction with the C
            if $dx = C$ then
                $unsat \leftarrow unsat(algo_x, algo_y) + 1$
            else
                $dy \neq C$
                $sat \leftarrow sat(algo_x, algo_y) + 1$
            end if
        end if
    until !EOF
end function
$(t_y : algo_x \rightarrow algo_y) \leftarrow \frac{sat(algo_x,algo_y)}{sat(algo_x,algo_y)+unsat(algo_x,algo_y)}$
```

is calculated to address this problem. The local trusts on an algorithm from all the other algorithms in the system are summed to calculate the recommended trust. The recommended trust value represents the combine trust of all algorithms in the system on that particular algorithm. If the set of all algorithms is $S = \{algo_0, algo_1, algo_2, \ldots, algo_n\}$ then we may have a two subsets $S' = \{algo_0\}$ and $S'' = \{algo_1, algo_2, \ldots, algo_n\}$. The subset S'' is having all the algorithms in the system as members except the algorithm $algo_0$ for which the RT is calculated. The algorithm $algo_0$ is the member of the subset S'. The RT is calculated by using the Equation (9.1).

$$RT_y \leftarrow \sum_{n=1}^{n} (t_y : algo_n \rightarrow algo_y) \qquad \forall algo_n \in S'' \tag{9.1}$$

Global Trust Calculation

The *RT* varies from algorithm to algorithm and may not be compared on the similar scale. Consequently, *RT* of an algorithm is normalized to obtain the global trust of that particular algorithm. The term normalization represents distinct, but related meanings in different contexts. The basic purpose of normalization is to convert the different values on a notionally standard scale to compare them equally with each other. The normalized *GT* value lies in the interval of the [0-1] and is calculated by using the Equation (9.2):

$$GT_y \leftarrow \frac{RT_y}{\sqrt{\sum_{n=1}^{n} RT_n^2}} \tag{9.2}$$

Veto Decision

The calculated *GT* value is used for deciding a veto for the prediction of a set of algorithms by another algorithm or set of other algorithms. Suppose a system in which seven algorithms are participating for predicting the class (benign or malicious) of an instance that belongs to the malicious class. A subset M of four algorithms in the system predicted the class of the instance as benign, and a subset V of three algorithms predicted the class of the instance as malicious. The mean of both groups is calculated. If the mean of V is greater than the mean of M, the V can veto the decision of the M and the outcome will be the prediction of the V.

However, for this experiment, there is a change in the veto decision function due to less algorithms. The change is explained as following. There are three algorithms in the system, i.e., $algo_x$, $algo_y$, and $algo_z$. If two algorithms, i.e., $algo_x$, and $algo_z$ classify the instance as a benign and only one algorithm, i.e., the $algo_y$ classify the instance as a malware; the $algo_y$ can veto according to Equation (9.3). The change in the prediction strategy is to reduce the random decision errors.

$$Veto : GT_y \geq \frac{GT_x + GT_z}{2} \tag{9.3}$$

9.6.2 Results and Discussion

The classifiers combination to form an ensemble can be divided roughly into two categories, i.e., multiclassifier and multirepresentation [115]. In the multiclassifier approach, a set of classifiers is trained on the same representation of the data. In the multirepresentation approach, different classifiers are trained on the multiple representations. On the basis of the experimental results presented in the Section 9.5.7, the trust-based veto classifier is applied only for combining the inductive biases of several algorithms trained on the different representations. The experimental results are shown in the Table 9.2. Experimental results indicate two issues. First, the low TN of the veto classifier that leads to a high FP; the low TN is because of veto classifier's skew towards malware. Second, the low TP of the majority voting that leads to a high FN; the low TP is because of the majority voting's skew towards benign programs. The trust-based veto classifier performs better than the veto classifier in terms of TN and reduces the FP. The trust-based veto classifier is also better than the majority voting in terms of TP and reduces the FN. However, TP of trust-based veto classifier is better than the majority voting and less than the veto classifier. Recall of the trust-based veto classifier is better than the majority voting and less than the veto voting. In terms of F-Measure, the difference between the values of f-measure of the majority voting and the trust-based veto classifier is minimal, so one can argue that the trust-based veto classifier is an optimal choice for the malware detection due to inherited skewness towards the malware. The majority voting and the veto classifier are computationally inexpensive as the prediction from each algorithm is counted for the outcome. In trust-based veto classifier, each algorithm evaluates the trust and maintains the trust info locally in a trust table without significantly increasing the processing overhead; however, the storage requirement is higher than the majority voting and the veto classifier as they do not store any information. The locally stored trust information is provided to the system for the decision purpose, when required. Trust-based veto classifier provides a direct experience of the trust on each algorithm. Due to direct experience, there is no central authority for maintaining the trust information, which makes the proposed algorithm a self-policing algorithm.

One property mentioned in the voting rule set is anonymity (see section 9.3.1). The trust-based veto classifier maintains the anonymity property as all votes are treated equally and no vote is discarded. However, the veto property is not followed as it is mentioned in section 9.3.1 for the trust-based veto classifier experiment. In the changed veto decision strategy, a single algorithm indicating the instance as the malware cannot affect the outcome of the detection task. Now for the veto, algorithm or set of algorithms need to meet certain criteria, which reduced the chances of errors and terminate the prediction of weak learners. However, with the proposed strategy, there is a probability that the prediction of weak learner/s may be always ignored, if the trust on that particular algorithm is significantly less than the trust on other algorithms. Suppose a detection system with five algorithms where two algorithms are weak learner with significantly low trust levels. This particular group of algorithms may not veto the decision of all other algorithms for all the cases, even if the prediction was correct. However, the changes in the veto strategy will increase the robustness of trust-based classifier as any number of algorithms can compete for the veto decision with any number of algorithms.

To follow the veto strategy mentioned in the section 9.3.1, one alternative direction is to allocate the trust to each algorithm on the basis of predetermined criteria such as previous performance. The trust of all algorithms may be readjusted regularly on the basis of prediction performance. All the algorithms vote for the decision. For the veto decision, when an algorithm predicts the instance as malware the trust of that particular algorithm can be compared with a specific threshold to obtain the final decision. However, it is worth to note that veto strategy does not perform as expected for the encrypted malware, i.e., malware with the encrypted malicious routine. The encrypted part of the malware cannot be disassembled to obtain accurate instruction sequences or byte code. The presence of encryption in a file can be considered as the indication of the malicious behavior. The encrypted malware can be decrypted or executed to decrypt in a controlled environment to obtain the data files. The data files can be further disassembled to extract instruction sequences or byte code.

9.7 Conclusion and Future Work

There are a several strategies to obtain the result of an ensemble such as the majority voting and the veto voting. However, it is not investigated which decision strategy is optimal for the malware detection. Most of the researchers have used the majority voting for the malware detection. A veto-based classification was proposed that was able to predict about malware better than the majority voting. A series of experiments with *n*-gram data sets, generated from different strategies, were performed. A recent threat, i.e., scareware was used as malware representation. The results indicated that the proposed model reduced the number of false negatives (malware detected as legitimate application), however, the false positive of proposed model was very high. The decision strategy of proposed model was improved, i.e., trust-based veto classifier. The experimental results indicated that the improved classifier perform better than the previous approach in terms of the false positive rate. The proposed trust-based veto classifier performed better in the recall than the majority voting. However, for the composite measure F1, the majority voting was slightly better than the trusted-veto classifier and the trusted veto classifier was better than the veto voting. The experimental results also indicated the suitability of each voting scheme for detecting a particular class of software. For the future work, the aim is to further improve the proposed model in two different directions, i.e., improvement in the selection of classifiers for the optimal results, and parameter tuning of the selected classifiers. The proposed model will also be tested for the detection of different types of malware and for the multi-class prediction.

Bibliography

[1] William Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, 4th edition, 2011.

[2] Lynn Greiner. The new face of malware. *netWorker*, 10(4):11–13, 2006.

[3] Rehan Shams, Muhammad Farhan, Sajid Ahmed Khan, and Fahad Hashmi. Comparing anti-spyware products – a different approach. In *Proceedings of the 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, volume 1, pages 75–80, 2011.

[4] Erin Egan and Tim Jucovy. Building a better filter: How to create a safer internet and avoid the litigation trap. *IEEE Security and Privacy*, 4(3):37–44, 2006.

[5] Ming-Wei Wu, Yi-Min Wang, Sy-Yen Kuo, and Yennun Huang. Self-healing spyware: Detection, and remediation. *IEEE Transactions on Reliability*, 56(4):588–596, 2007.

[6] Madhusudhanan Chandrasekaran, Vidyaraman Sankaranarayanan, and Shambhu J. Upadhyaya. SpyCon: Emulating user activities to detect evasive spyware. In *Proceedings of the International Performance Computing and Communications Conference*, pages 502–509, 2007.

[7] Alan Westin. *Privacy and Freedom*. Atheneum, 1st edition, 1967.

[8] Thomas F. Stafford and Andrew Urbaczewski. Spyware: The ghost in the machine. *Communications of the Association for Information Systems*, 14(1), 2004.

[9] Paul McFedries. Technically speaking: The spyware nightmare. *IEEE Spectrum*, 42(8):72–72, 2005.

[10] Charles D. Curran. Combating spam, spyware, and other desktop intrusions: Legal considerations in operating trusted intermediary technologies. *IEEE Security and Privacy*, 4(3):45–51, 2006.

[11] Lorenzo Martignoni, Elizabeth Stinson, Matt Fredrikson, Somesh Jha, and John C. Mitchell. A layered architecture for detecting malicious behaviors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, pages 78–97, 2008.

[12] Nathanael R. Paul. *Disk-level behavioral malware detection*. PhD dissertation, University of Virginia, USA, 2008.

[13] Paul Piccard and Jeremy Faircloth. *Combating Spyware in the Enterprise*. Syngress Publishing, 2006.

[14] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[15] Thomas M. Chen and Jean marc Robert. The evolution of viruses and worms. In *Statistical Methods in Computer*, 2004.

[16] Zhen Li, Qi Liao, and Aaron Striegel. Botnet economics: Uncertainty matters. In *Managing Information Risk and the Economics of Security*, pages 245–267. Springer, 2009.

[17] Cong Zheng, Lansheng Han, Jihang Ye, Mengsong Zou, and Qiwen Liu. A fuzzy comprehensive evaluation model for harms of computer virus. In *Proceedings of the 6th International Conference on Mobile Adhoc and Sensor Systems*, pages 708–713, 2009.

[18] Martin Boldt and Bengt Carlsson. Privacy-invasive software and preventive mechanisms. In *Proceedings of the International Conference on Systems and Networks Communication*, pages 21–27. IEEE Computer Society, 2006.

[19] Alvin Loh, Angela K. Butcher, Jason Garms, Kalid M. Azad, Marc E. Seinfeld, Paul J. Bryan, and Sterling M. Reasor. System and method for identifying and removing potentially unwanted software, 2006. EP Patent 1,708,115.

[20] Raja Khurram Shahzad, Syed Imran Haider, and Niklas Lavesson. Detection of spyware by mining executable files. In *Proceedings of the 5th International Conference on Availability, Reliability, and Security*, pages 295–302. IEEE Computer Society, 2010.

[21] Richard H. Stern. FTC cracks down on spyware and pc hijacking, but not true lies. *IEEE Micro*, 25(1):6–7, 2005.

[22] Geoff Shaw. Spyware & adware: the risks facing businesses. *Network Security*, 2003(9):12–14, 2003.

[23] Sarah Gordon. Fighting spyware and adware in the enterprise. *Information Systems Security*, 14(3):14–17, 2005.

[24] Raja Khurram Shahzad and Niklas Lavesson. Detecting scareware by mining variable length instruction sequences. In *Proceedings of the 10th Annual Information Security South Africa Conference*, pages 1–8. IEEE Press, 2011.

[25] Niklas Lavesson, Paul Davidsson, Martin Boldt, and Andreas Jacobsson. *Spyware Prevention by Classifying End User License Agreements*, volume 134. Springer, 2008.

[26] Yanfang Ye, Tao Li, Qingshan Jiang, and Youyu Wang. Cimds: adapting postprocessing techniques of associative classification for malware detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(3):298–307, 2010.

[27] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[28] Adrian Stepan. Improving proactive detection of packed malware. *Virus Bulletin*, pages 11–13, 2006.

[29] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malcode detection using OPCODE representation. In *Proceedings of the 1st European Conference on Intelligence and Security Informatics*, pages 204–215. Springer, 2008.

[30] Oren Drori, Nicky Pappo, and Dan Yachan. New malware distribution methods threaten signature-based AV. *Virus Bulletin*, pages 9–11, 2005.

[31] Jose Nazario. *Defense and Detection Strategies against Internet Worms*. Artech House, Inc., 2003.

[32] Andreas Marx. Antivirus outbreak response testing and impact. *Virus Bulletin*, 2004.

[33] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, 2001.

[34] Moheeb Abu Rajab, Lucas Ballard, Panayiotis Mavrommatis, Niels Provos, and Xin Zhao. The nocebo effect on the web: an analysis of fake anti-virus distribution. In *Proceedings of the 3rd USENIX Conference on Large-scale exploits and Emergent threats: Botnets, Spyware, Worms, and more*, pages 3–3. USENIX Association, 2010.

[35] Aristotle. *The Complete Works of Aristotle: The Revised Oxford Translation*, volume 2. Princeton University Press, 1984.

[36] Samuel Warren and Louis D. Brandeis. The right to privacy. *Harvard Law Review*, 4(5), 1890.

[37] Batya Friedman, Edward Felten, and Lynette I. Millett. Informed Consent Online: A Conceptual Model and Design Principles. Technical report, University of Washington, 2003.

[38] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Inc., 3rd edition, 2011.

[39] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1 edition, 1997.

[40] Fred Cohen. Computational aspects of computer viruses. *Computers & Security*, 8(4):297–298, 1989.

[41] Fred Cohen. *Computational aspects of computer viruses*, chapter in Rogue programs: viruses, worms and Trojan horses, pages 324–355. Van Nostrand Reinhold, 1990.

[42] Jeffrey M. Voas, Jeffrery E. Payne, and Frederick B. Cohen. A model for detecting the existence of unknown computer viruses in real-time. In *Proceedings of 5th International Computer Virus & Security Conference*, 1992.

[43] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4:251–266, 2008.

[44] Peter J. Denning, editor. *Computers under attack: intruders, worms, and viruses*. ACM, 1990.

[45] Nwokedi Idike and Aditya P. Mathur. A survey of malware detection techniques. Technical report, Purdue University, 2007.

[46] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Pro-*

ceedings of the 18th Conference on USENIX security symposium, pages 351–366. USENIX Association, 2009.

[47] Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang. *Malware Detection (Advances in Information Security)*. Springer, 2006.

[48] Heng Yin. *Malware Detection and Analysis Via Layered Annotative Execution*. BiblioBazaar, 2011.

[49] Christian Kreibich and Jon Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *Proceedings of the 2nd Workshop on Hot Topics in Networks*, 2003.

[50] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *Recent Advances in Intrusion Detection*, volume 2820, pages 173–191. Springer, 2003.

[51] Carey Nachenberg. Generic exploit blocking. *Virus Bulletin*, 2005.

[52] Carey Nachenberg. Generic exploit blocking: Prevention, not cure. *Information Systems Audit and Control Association*, 2005.

[53] Kent Griffin, Scott Schneider, Xin Hu, and Tzi-Cker Chiueh. Automatic generation of string signatures for malware detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, pages 101–120. Springer, 2009.

[54] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 213–223. ACM, 2005.

[55] Cristian Cadar and Dawson Engler. Execution generated test cases: how to make systems code crash itself. In *Proceedings of the 12th International Conference on Model Checking Software*, pages 2–23. Springer, 2005.

[56] Jeffrey O. Kephart and William C. Arnold. Automatic extraction of computer virus signatures. *4th Virus Bulletin International Conference*, pages 178–184, 1994.

[57] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A State-of-the-Art survey. *Information Security Technical Report*, 14:16–29, 2009.

[58] William B Cavnar and John M Trenkle. N-Gram-Based text categorization. *Proceedings Of 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[59] ChengXiang Zhai. *Statistical Language Models for Information Retrieval*. Morgan & Claypool, 2009.

[60] Vlado Keselj, Fuchun Peng, Nick Cercone, and Calvin Thomas. N-gram-based author profiles for authorship attribution. *Computational Linguistics*, 3, 2003.

[61] Gerald Tesauro, Jeffrey O. Kephart, and Gregory B. Sorkin. A neural network virus detector. *In Proceedings of the IBM Security ITS*, 1994.

[62] Gerald Tesauro, Jeffrey O. Kephart, and Gregory B. Sorkin. Neural networks for computer virus recognition. *In IEEE Expert*, pages 5–6, 1996.

[63] William Arnold and Gerald Tesauro. Automatically generated win32 heuristic virus detection. In *Proceedings of the 2000 International Virus Bulletin Conference*, 2000.

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

[65] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. N-gram based detection of new malicious code. In *Proceedings*

of the 28th Annual International Computer Software and Applications Conference, pages 41–52. IEEE Computer Society, 2004.

[66] Olivier Henchiri and Nathalie Japkowicz. A feature selection and evaluation scheme for computer virus detection. In *Proceedings of the 6th International Conference on Data Mining*, pages 891–895, 2006.

[67] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. Unknown malcode detection via text categorization and the imbalance problem. In *Proceedings of the International Conference on Intelligence and Security Informatics*, pages 156–61. IEEE, 2008.

[68] Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. Detecting trojans using data mining techniques. In *Proceedings of the Internationa Multi-Topic Conference*, volume 20, pages 400–411. Springer, 2008.

[69] Anthonius Sulaiman, K. Ramamoorthy, Srinivas Mukkamala, and Andrew H. Sung. Disassembled code analyzer for malware (DCAM). In *Proceedings of the International Conference on Information Reuse and Integration*, pages 398–403, 2005.

[70] Raja Khurram Shahzad, Niklas Lavesson, and Henric Johnson. Accurate adware detection using opcode sequence extraction. In *Proceedings of the 6th International Conference on Availability, Reliability, and Security*, pages 189–195. IEEE Press, 2011.

[71] Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. Detecting internet worms using data mining techniques. *Journal of Systemics, Cybernetics and Informatics*, 6(6):48–53, 2008.

[72] Lili Bai, Jianmin Pang, Yichi Zhang, Wen Fu, and JiaFeng Zhu. Detecting malicious behavior using critical api-calling graph matching. In *Proceedings of 1st International Conference on Information Science and Engineering*, pages 1716–1719, 2009.

[73] Shanhu Shang, Ning Zheng, Jian Xu, Ming Xu, and Haiping Zhang. Detecting malware variants via function-call graph similarity. In *Proceedings of 5th International Conference on Malicious and Unwanted Software*, pages 113–120, 2010.

[74] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *Journal in Computer Virology*, 7:233–245, 2011.

[75] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. Wiley-IEEE Press, 2nd edition, 2011.

[76] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall (CRC Press), 4 edition, 2007.

[77] William R Shadish, Thomas D Cook, and Donald T Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin, 2001.

[78] Colin Robson. *Real World Research*. John Wiley & Sons Inc, 2011.

[79] Donald T. Campbell and Julian C. Stanley. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin Company, 1966.

[80] John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications Ltd., 3rd edition, 2008.

[81] Thomas D. Cook and Donald T. Campbell. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin, 1979.

[82] Ming-Wei Wu and Sy-Yen Kuo. Examining web-based spyware invasion with stateful behavior monitoring. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, pages 275–281. IEEE Computer Society, 2007.

[83] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(9-19), 1993.

[84] Narges Arastouie and Mohammad Reza Razzazi. Hunter: An anti spyware for windows operating system. In *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–5, 2008.

[85] Terry Bollinger. Software in the year 2010. *IT Professional*, 6(6):11–15, 2004.

[86] Qing Hu and Tamara Dinev. Is spyware an internet nuisance or public menace? *Communications of the ACM*, 48(8):61–66, 2005.

[87] Wes Ames. Understanding spyware: Risk and response. *IT Professional*, 6(5):25–29, 2004.

[88] Cumhur Doruk Bozagac. Application of data mining based malicious code detection techniques for detecting new spyware. *White paper, Bilkent University*, 2005.

[89] Ming-Wei Wu, Yennun Huang, Yi-Min Wang, and Sy-Yen Kuo. A stateful approach to spyware detection and removal. In *Proceedings of the 12th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 173–182. IEEE Computer Society, 2006.

[90] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005.

[91] William W. Cohen. Fast effective rule induction. In *Proceedings of 12th International Conference on Machine Learning*, pages 115–23. Morgan Kaufmann Publishers, 1995.

[92] Yuval Elovici, Asaf Shabtai, Robert Moskovitch, Gil Tahan, and Chanan Glezer. Applying machine learning techniques for detection of malicious code in network traffic. In *KI 2007: Advances in Artificial Intelligence*, volume 4667, pages 44–50. Springer, 2007.

[93] Jau-Hwang Wang, Peter S. Deng, Yi-Shen Fan, Li-Jing Jaw, and Yu-Ching Liu. Virus detection using data mining techinques. In *Proceedings of the IEEE 37th Annual International Carnahan Conference on Security Technology*, pages 71–76, 2003.

[94] Niklas Lavesson, Martin Boldt, Paul Davidsson, and Andreas Jacobsson. Learning to detect spyware using end user license agreements. *Knowledge and Information Systems*, 26(2):285–307, 2011.

[95] D. Krishna Sandeep Reddy, Subrat Kumar Dash, and Arun K. Pujari. New malicious code detection using variable length n-grams. In *Information Systems Security*, volume 4332, pages 276–288. Springer, 2006.

[96] Anders Isaksson, Mikael Wallman, Hanna Göransson, and Mats G. Gustafsson. Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, 29:1960–1965, 2008.

[97] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations*, 11:10–18, 2009.

[98] Foster J. Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann Publishers Inc., 1998.

[99] Graham Cluley. Malware trends: Sizing up the malware threat - key malware trends for 2010. *Network Security*, 2010(4):8–10, 2010.

[100] Marco Cova, Corrado Leita, Olivier Thonnard, Angelos D. Keromytis, and Marc Dacier. An analysis of rogue av campaigns. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, pages 442–463. Springer, 2010.

[101] Luis Corrons. The business of rogueware. In *Web Application Security*, volume 72, pages 7–7. Springer, 2010.

[102] Shlomo Dolev and Nir Tzachar. Malware signature builder and detection for executable code, 2010. EP Patent 2,189,920.

[103] Gerard M. Salton, Andrew Wong, and ChungShu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, 1975.

[104] Mondelle Simeon and Robert Hilderman. Categorical proportional difference: A feature selection method for text categorization. In *Proceedings of the 7th Australasian Data Mining Conference*, volume 87, pages 201–208. ACS, 2008.

[105] William W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the 13th national Conference on Artificial intelligence*, pages 709–716. AAAI Press, 1996.

[106] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[107] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 208:98–112, 1999.

[108] Cao Feng and Donald Michie. *Machine learning of rules and trees*, pages 50–83. Ellis Horwood, 1994.

[109] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[110] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[111] E. Eugene Schultz. Pandora's box: spyware, adware, autoexecution, and ngscb. *Computers & Security*, 22(5):366–367, 2003.

[112] Juraj Malcho. Is there a lawyer in the lab ? In *Proceedings of the 19th Virus Bulletin International Conference*, 2009.

[113] Sam Scott and Stan Matwin. Feature engineering for text classification. In *Proceedings of the 16th International Conference on Machine Learning*, pages 379–388. Morgan Kaufmann Publishers Inc., 1999.

[114] Yanmin Sun, Mohamed S. Kamel, and Andrew K. C. Wong. Empirical study on weighted voting multiple classifiers. In *Proceedings of the 3rd International Conference on Advances in Pattern Recognition and Data Mining*, pages 335–344, 2005.

[115] Mohamed S. Kamel and Nayer M. Wanas. Data dependence in combining classifiers. In *Proceedings of the 4th International Conference on Multiple Classifier Systems*, pages 1–14. Springer, 2003.

[116] Louisa Lam and Simon. Y. Suen. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 27(5):553–568, 1997.

[117] Yu-An Sun and Christopher R. Dance. When majority voting fails: Comparing quality assurance methods for noisy human computation environment. *Computing Research Repository*, 1204.3516, 2012.

[118] Raja Khurram Shahzad and Niklas Lavesson. Veto-based malware detection. In *Proceedings of 7th International Conference on Availability, Reliability, and Security*, pages 47–54. IEEE Press, 2012.

[119] Chao Ren, Jian-Feng Yan, and Zhan-Huai Li. Improved ensemble learning in fault diagnosis system. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 1, pages 54–60, 2009.

[120] Ludmila I. Kuncheva. Diversity in multiple classifier systems. *Information Fusion*, 6(1):3–4, 2005.

[121] Tyrone W. A. Grandison. *Trust management for internet applications*. PhD thesis, Imperial College, 2003.

[122] Alexander Gepperth. Object detection and feature base learning with sparse convolutional neural networks. In *Artificial Neural Networks in Pattern Recognition*, pages 221–232, 2006.

[123] Domonkos Tikk, Zsolt Tivadar Kardkovács, and Ferenc. P Szidarovszky. Voting with a parameterized veto strategy: Solving the KDD cup 2006 problem by means of a classifier committee. *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations*, 8(2):53–62, 2006.

[124] Roman Kern, Christin Seifert, Mario Zechner, and Michael Granitzer. Vote/veto meta-classifier for authorship identification - notebook for pan at clef 2011. In *Proceedings of the Conference on Multilingual and Multimodal Information Access Evaluation*, 2011.

[125] Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, 53(4):1483–1494, 2009.

[126] Boyun Zhang, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. Malicious codes detection based on ensemble learning. In *Autonomic and Trusted Computing*, volume 4610, pages 468–477. Springer, 2007.

[127] Thomas Beth, Malte Borcherding, and Birgit Klein. Valuation of trust in open networks. In *Proceedings of the 3rd European Symposium on Research in Computer Security*, pages 3–18, 1994.

[128] Michael K. Reiter and Stuart G. Stubblebine. Path Independence for Authentication in Large-Scale Systems. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 57–66, 1997.

[129] Raph Levien. Attack resistant trust metrics. Technical report, UC Berkeley, 2004.

[130] Raph Levien. Attack-resistant trust metrics. In *Computing with Social Trust*, pages 121–132. Springer, 2009.

[131] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in

p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, pages 640–651, 2003.

[132] Zoë Abrams, Robert McGrew, and Serge Plotkin. A non-manipulable trust system based on eigentrust. *Special Interest Group on Electronic Commerce Exchanges*, 5(4):21–30, 2005.

[133] Xia Li, Jill Slay, and Shaokai Yu. Evaluating trust in mobile ad hoc networks. In *Proceedings of the International Conference on Computational Intelligence and Security Workshops*, 2005.

[134] Xin Liu, Gilles Trédan, and Anwitaman Datta. A generic trust framework for large-scale open systems using machine learning. *Computing Research Repository*, 1103.0086, 2011.

[135] Sholom M. Weiss, Chidanand Apte, Fred J. Damerau, David E. Johnson, Frank J. Oles, Thilo Goetz, and Thomas Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems and their Applications*, 14(4):63–69, 1999.

[136] Giorgio Giacinto and Fabio Roli. Methods for dynamic classifier selection. In *Proceedings of International Conference on Image Analysis and Processing*, pages 659–664, 1999.

[137] Alixandre Santana, Rodrigo G. F. Soares, Anne M. P. Canuto, and Marcílio Carlos Pereira de Souto. A dynamic classifier selection method to build ensembles using accuracy and diversity. In *Proceedings of the 9th Brazilian Symposium on Neural Networks*, pages 36–41, 2006.

[138] Fatima Talib. Computational aspects of voting: A literature survey. *Masterś Thesis, Rochester Institute of Technology*, 2007.

[139] Herve Moulin. Voting with proportional veto power. *Econometrica*, 50(1):145–62, 1982.

[140] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. In *Information Processing and Managment*, pages 513–523, 1988.

[141] Raymond H. Walpole, Ronald E.and Myers and Sharon L. Myers. *Probability & Statistics for Engineers & Scientists*. Prentice Hall, 2012.

[142] J. Franke, Louisa Lam, Raymond Legault, Christine P. Nadal, and Ching Y. Suen. Experiments with the CENPARMI data base combining different classification approaches. In *Proceedings of 3rd International Workshop Frontiers Handwriting Recognition*, pages 305–311, 1993.

[143] Ahmad F. R. Rahman, Hassan Alam, and Michael C. Fairhurst. Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations. In *Proceedings of the 5th International Workshop on Document Analysis Systems*, volume 2423, pages 167–178. Springer, 2002.

[144] Dymitr Ruta and Gabrys Bogdan. Classifier selection for majority voting. *Information Fusion*, 6(1):63–81, 2005.

ABSTRACT

Malicious software authors have shifted their focus from illegal and clearly malicious software to potentially unwanted programs (PUPs) to earn revenue. PUPs blur the border between legitimate and illegitimate programs and thus fall within a grey zone. Existing anti-virus and anti-spyware software are in many instances unable to detect previously unseen or zero-day attacks and separate PUPs from legitimate software. Many tools also require frequent updates to be effective. By predicting the class of a particular piece of software, users can get support before taking the decision to install the software. This Licentiate thesis introduces approaches to distinguish PUP from legitimate software based on the supervised learning of file features represented as *n*-grams.

The overall research method applied in this thesis is experiments. For these experiments, malicious software applications were obtained from anti-malware industrial partners. The legitimate software applications were collected from various online repositories. The general steps of supervised learning, from data preparation (*n*-gram generation) to evaluation were, followed. Different data representations, such as byte codes and operation codes, with different configurations, such as fixed-size, variable-length, and overlap, were investigated to generate different *n*-gram sizes. The experimental variables were controlled to measure the correlation between *n*-gram size, the number of features required for optimal training, and classifier performance.

The thesis results suggest that, despite the subtle difference between legitimate software and PUP, this type of software can be classified accurately with a low false positive and false negative rate. The thesis results further suggest an optimal size of operation code-based n-grams for data representation. Finally, the results indicate that classification accuracy can be increased by using a customized ensemble learner who makes use of multiple representations of the data set. The investigated approaches can be implemented as a software tool with a less frequently required update in comparison to existing commercial tools.

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

ISSN 1650-2140
ISBN: 978-91-7295-247-8

2013:02