Forensic Identification and Detection of Hidden and Obfuscated Malware

Mamoun Atef Alazab

Bachelor degree of Computer Science

Higher Diploma degree of Computer Science

Master degree of Information Technology

This thesis is submitted in total fulfilment

of the requirements for the degree of

Doctor of Philosophy

School of Science, Information Technology and Engineering

University of Ballarat

PO Box 663

University Drive, Mount Helen

Ballarat, Victoria 3353

Australia

January 2012

Principal Supervisor:

Dr Sitalakshmi Venkatraman

Associate Supervisor:

Associate Professor Paul Watters

Statement of Authorship

Except where explicit reference is made in the text of the thesis, this thesis contains no material published elsewhere or extracted in whole or in part from a thesis by which I have qualified for or been awarded another degree or diploma. No other person's work has been relied upon or used without due acknowledgement in the main text and bibliography of the thesis.

Signed: _____

Name: _____

Dated: _____

Forensic Identification and Detection of Hidden and Obfuscated Malware

Mamoun Alazab

January 2012

Dedication

To my parents, who gave me what I need to get here.

Your love and support have given me the biggest strength.

Acknowledgements

During the duration of my PhD study I was blessed with a combination of excellent supervision, institutional assistance, inspirational colleagues, industry encouragement and a supportive family. I have been fortunate to have enjoyed this level of support throughout this endeavour. It is not possible to acknowledge everyone who has made a contribution to this study, but special thanks go to those who have given me generous support and encouragement during my time as a PhD student and during the process of writing this dissertation, who have supported me greatly during my PhD. It has been a difficult time and without them I could not have done it.

First, special thanks to my wonderful parents: my dad and my mum who have provided much support and encouragements over the years.

Very special recognition is due to my principal supervisor, Dr Sitalakshmi Venkatraman. Without her encouragement, mentorship and her continuous support, I doubt I would find myself here. I am indebted to Dr Venkatraman for her feedback, enthusiasm, editing skills, wisdom, insight and expertise in the field of computer security, without which this dissertation would undoubtedly be a lesser contribution to knowledge. I also thank her for her tireless support, motivation and encouragement throughout my studies, for the many hours she devoted in supervising my work, and for setting an exceptional role model in research excellence. I also would like to thank her for her 5 P's (Plan, Prepare, Passion, Persistence and Patience) and for reminding me when they were needed.

My biggest thanks must be given to my Associate Professor Paul Watters for his unconditional support, respect and belief in me. I appreciate his vast profound knowledge and skills in many areas. I am greatly indebted for his valuable instructions and suggestions in my research work. I have learnt from him a lot not only about academic studies, but also the professional ethics.

This research has been supported by IBM, Westpac, the Victorian Government and the Australian Federal Police, the shared communications between these sectors has been invaluable and has offered insight where blindness would have ensued. The team at Westpac in particular have been invaluable with their knowledge, insight and feedback, and have provided me with lots of opportunities to improve as a researcher and as a professional. I would like to express my heartfelt gratitude to Simon Brown of Westpac for his valuable ideas, suggestions, constant encouragement and guidance during my research work. Without such inputs, this thesis would not have reached the required practical applicability that has been achieved, which is an important aspect in cybercrime research.

I would also like to thank numerous colleagues, who offered advice and constant encouragement, I would like to acknowledge the helpful support in particular: Kylie Turville, Rosemary Torney, Oarabile Maruatona, and Mofakharul Islam, for their patience, support, guidance and being a great friend throughout the last 3 years of my study and I hope to have the opportunity to continue to interact with all of them in the future. Special thanks to my colleague Robert Layton, who gave me support and assist at important times as research assistance during data collection and data analysis.

I would like to thank my two brothers Ammar Alazab and Moutaz Alazab for being with me in this long journey and joining me in Australia to be as a family. Also, I would like to thank all my siblings; Abed Almajed Alazab, Omar Alazab, my only sister Remah Alazab, who have all given me support throughout my life, have helped to make this possible, and have ultimately made the end result mean much more than just obtaining a degree.

Also, I would like to thank my patient and wonderful partner, Penny Butler, who copyedited my first PhD paper in this dissertation and proofread some of my emails, for her deep understanding, profound encouragements, unlimited support, constant belief, love, help, and patience.

The wonderful staff of the University of Ballarat, those within the Research and Graduate Studies Office, in particular Diane Clingin and Elanor Mahon, and the School of Science, Information Technology and Engineering, in particular Prof. John Yearwood (Dean) with administrative support from Maxine Kingston, Yve Rowe and Rebecca Davis. The support offered by the Internet Commerce Security Laboratory team has been invaluable to me.

Finally, I would also like to take this opportunity to thank the examiners for their time and devotion in regarding my PhD thesis.

List of Publications

The following papers, I author and co-author, which have been previously published or are currently in submission, are reprinted in this dissertation with the full permission of all co-authors of the papers:

JOURNAL PUBLICATIONS

23- Mamoun Alazab, Shamsul Huda, Paul Watters, Sitalakshmi Venkataraman and John Yearwood "A novel Malware detection using hybrid Wrapper-Filter approach and op-code frequency statistics based on extended x86 IA-32 binary assembly instructions", (submitted).

22- Salah Al-Hyari, Moutaz Alazab, Sitalakshmi Venkatraman, Mamoun Alazab, and Ammar Alazab "Six Sigma Approach to Improve Quality in e-Services – An Empirical Study in Jordan" (Accepted) IGI Global, The International Journal of Electronic Government Research (IJEGR), 2011.

21- Salah Al-Hyari, Moutaz Alazab, Sitalakshmi Venkatraman, Mamoun Alazab, and Ammar Alazab "Performance Evaluation of E-Government Services Using Balance Scorecard An Empirical Study Jordan E-Government" (Accepted) Benchmarking: an International Journa, Taylor & Francis, Local Government Studies, 2011.

20- Mamoun Alazab, Mohammad Alkadiri, and Sitalakshmi Venkatraman "Multivariate Logistic regression model for Malware anomaly detection based on operation code instructions", (Submitted) Journal of Computing, 2011.

19- Said Elaiwat, Ammar Alazab, Sitalakshmi Venkatraman and Mamoun Alazab "Applying Genetic Algorithm for Optimizing Broadcasting Process in Ad-hoc Network", The International Journal of Recent Trends in Engineering and Technology, The Association of

Computer Electronics and Electrical Engineers , ISSN: 1797-9617, Vol.4, No.1, 2010, pp. 68 – 72.

18- Mamoun Alazab, Sitalakshmi Venkataraman and Paul Watters “Effective digital forensic analysis of the NTFS disk image”, Ubiquitous Computing and Communication Journal, ISSN 1994-4608, Vol. 4, No. 3, 2009, pp. 551- 558.

BOOK CHAPTERS

17- Mamoun Alazab, Paul Watters, Sitalakshmi Venkatraman, and Moutaz Alazab “Malware Forensics: The Art of Detecting Hidden Maliciousness” Book Title: IT Security Governance Innovations: Theory and Research, (Accepted to publish), IGI Global, 2011.

16- Ammar Alazab, Sitalakshmi Venkatraman, Jemal Abawajy, and Mamoun Alazab "An Optimal Transportation Routing Approach using GIS-based Dynamic Traffic Flows", Book Title: Management Technology and Applications Proceedings of the International Conference on. In Rawani, A. & Zhang C. (Eds), ISBN 978-981-08-6884-0, Research Publishing Services, 2010, pp. 168 – 174.

15- Mofakharul Islam, Sitalakshmi Venkatraman and Mamoun Alazab “Stochastic Model Based Approach for Biometric Identification”, Book Title: Technological Developments in Networking, Education and Automation. In K. Elleithy, T. Sobh, M. Iskander, V. Kapila, M. A. Karim & A. Mahmood (Eds.) Engineering, ISBN: 978-90-481-9151-2, Springer Netherlands, 2010, pp. 303-308.

14- Mamoun Alazab, Mofakharul Islam, Sitalakshmi Venkatraman “Towards Automatic Image Segmentation Using Optimised Region Growing Technique”, Book Title: AI 2009: Advances in Artificial Intelligence. In A. Nicholson & X. Li (Eds.) Lecture Notes in

Computer Science, ISBN: 978-3-642-10438-1, Springer-Verlag Berlin Heidelberg, 2009, Vol.5866, pp. 131-139.

CONFERENCE PUBLICATIONS

13- Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab "Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures" Ninth Australasian Data Mining Conference: AusDM 2011, CRPIT series, 1-2 December 2011, Ballarat, Victoria, Australia.

12- Mamoun Alazab, Paul Watters, Sitalakshmi Venkatraman, Ammar Alazab, and Moutaz Alazab "Cybercrime: Current Trends of Malware Threats" International Conference in Global Security Safety and Sustainability / International Conference on e-Democracy, CCIS series from Springer. 24-26 August, 2011, Thessaloniki, Greece.

11- Said Elaiwat, Ammar Alazab, Sitalakshmi Venkatraman and Mamoun Alazab "GOM: New Genetic Optimizing Model for Broadcasting Tree in MANET", Proceedings of The IEEE second International Conference on Computer Technology and Development, IEEE Computer Society, ISBN: 978-1-4244-8844-5, 2010, 2-4 November, Cario, Egypt, pp. 477-481.

10- Mamoun Alazab, Sitalakshmi Venkataraman and Paul Watters "Towards Understanding Malware Behaviour by the Extraction of API Calls", Cybercrime and Trustworthy Computing, Workshop, IEEE Computer Society, 2010, 0, pp. 52-59.

9- Mamoun Alazab, Robert Layton, Sitalakshmi Venkataraman and Paul Watters "Malware Detection Based on Structural and Behavioural Features of API calls", Proceedings The 1st International Cyber Resilience conference, School of Computer and Information Science,

Security Research Centre, Edith Cowan University, ISBN 978-0-7298-0690-9, 2010, 23-24 August, Perth, WA, pp. 1-10.

8- Mohammad Alkhaleefah, Mahmoud Alkhawaldeh, Sitalakshmi Venkatraman, and Mamoun Alazab "Towards understanding and improving e-government strategies in Jordan", International Conference on e-Commerce, e-Business and e-Service, World Academy of Science, Engineering and Technology, ISSN: 2070-3740, Issue 66, 2010, 28-30 June, Paris, France, pp. 1871-1877.

7- Mamoun Alazab, Sitalakshmi Venkataraman and Paul Watters "Digital forensic techniques for static analysis of NTFS images", The Fourth International Conference on Information Technology, ISBN 9957-8583-0-0, 2009, Amman, Jordan.

ABSTRACTS & PRESENTATIONS

6- Mamoun Alazab "Static analysis for Anomaly and Similarity based detection", Annual Research Conference, University of Ballarat, Ballarat, VIC, 2011, 3- 4 November, pp. 87.

5- Mamoun Alazab "Static Analysis of Obfuscated Malware", Annual Research Conference, University of Ballarat, Ballarat, VIC, 2010, 3 November.

4- Mamoun Alazab "Detection of Malware based on Behaviour Identification", University of Ballarat, Three Minute Thesis Competition, July 2010.

3- Mamoun Alazab "Forensic identification of hidden malware in Physical", University of Ballarat, 2nd Cybercrime and Trustworthy Computing Workshop, July 2010.

2- Mamoun Alazab "Investigation techniques for static analysis of NTFS file system images", Annual Research Conference, University of Ballarat, Ballarat, VIC, 2009, 11 November, pp. 23.

1- Mamoun Alazab "Obfuscated Malware Detection", HCSNet Workshop on Statistical Parametric Mapping for Functional Magnetic Resonance Imaging, 2009.

Abstract of the Thesis

Forensic Identification and Detection of Hidden and Obfuscated Malware

Mamoun Alazab
Doctor of Philosophy in Information Technology
School of Science, Information Technology and Engineering
University of Ballarat,
Ballarat, Victoria
Australia

The revolution in online criminal activities and malicious software (malware) has posed a serious challenge in malware forensics. Malicious attacks have become more organized and purposefully directed. With cybercrimes escalating to great heights in quantity as well as in sophistication and stealth, the main challenge is to detect hidden and obfuscated malware. Malware authors use a variety of obfuscation methods and specialized stealth techniques of information hiding to embed malicious code, to infect systems and to thwart any attempt to detect them, specifically with the use of commercially available anti-malware engines. This has led to the situation of zero-day attacks, where malware inflict systems even with existing security measures. The aim of this thesis is to address this situation by proposing a variety of novel digital forensic and data mining techniques to automatically detect hidden and obfuscated malware.

Anti-malware engines use signature matching to detect malware where signatures are generated by human experts by disassembling the file and selecting pieces of unique code. Such signature based detection works effectively with known malware but performs poorly with hidden or unknown malware. Code obfuscation techniques, such as

packers, polymorphism and metamorphism, are able to fool current detection techniques by modifying the parent code to produce offspring copies resulting in malware that has the same functionality, but with a different structure. These evasion techniques exploit the drawbacks of traditional malware detection methods, which take current malware structure and create a signature for detecting this malware in the future. However, obfuscation techniques aim to reduce vulnerability to any kind of static analysis to the determent of any reverse engineering process. Furthermore, malware can be hidden in file system slack space, inherent in NTFS file system based partitions, resulting in malware detection that even more difficult.

Security researchers and the anti-malware industry are facing a herculean task in extracting the payload, the de-obfuscated malware, and maliciousness hidden by these techniques. For effective and efficient solutions, this thesis moves away from the signature based detection to anomaly based detection. This thesis aims to provide solutions to the problem of detecting hidden and obfuscated malware through the following major contributions to the literature:

One – Propose a method for detecting malicious software that is hidden in NTFS slack space partitions. NTFS file system provides useful information leading towards identifying hidden malware and presentation of digital evidence for the court of law. Chapter 3 of thesis provides knowledge, methodology and discusses the analysis techniques used to successfully detect maliciousness in hidden data and hidden space, by investigating the NTFS file system boot sector.

Two – Extracting features from the obfuscated executables for reverse obfuscation is labor intensive and requires deep understanding of kernel and assembly programming. Develops fully automated system to extract two independent features, namely OP code and API function call features for finding the fingerprint of executable programs and for detection and differentiation of different files that are either malicious or benign.

Three – Propose anomaly based detection that focuses on the extended x86 IA-32 binary assembly instructions' frequency statistics, using i) Maximum Relevance (MR) filter heuristic, ii) Artificial Neural Net Input Gain Measurement Approximation (ANNIGMA), and iii) combination of MR and ANNIGMA (MR-ANNIGMA). Experimental results show that our frequency-statistics based approach achieves high accuracy ~96%.

Four – Propose similarity based detection of unknown malware using API function calls features, using various distance measures of vector models. Experimental analysis claims that our proposed method is an effective method to accurately differentiate malware from benign files and, more importantly, to detect obfuscated malware families.

Five – Investigate the employment of various robust supervised machine learning algorithms on API function call features. Experimental results achieved high true positive rate of ~ 99%, and low false positive rate of less than 2%, which has not been achieved in literature so far. This is much higher than the required commercial acceptance level indicating that our novel technique is a major leap forward in detecting zero-day malware.

In this thesis, we have demonstrated the robustness of our proposed techniques and automated system by testing it on a large dataset of 66,703 executable files in total,

consisting of 51,223 recent Malware samples collected over the period 2008-2011, with the remaining executables forming benign samples. Together, these contributions enable an effective and efficient forensic identification and detection of hidden and obfuscated malware.

Table of Contents

List of Figures

List of Tables

Chapter 1 : Introduction

SHERLOCK HOLMES*: "It is of the highest importance in the art of detection to be able to recognize out of a number of facts which are incidental and which vital. Otherwise your energy and attention must be dissipated instead of being concentrated."*

—Sir Arthur Conan Doyle, "The Adventure of the Reigate Squire,"

The Strand Magazine (1893)

1.1 Background

The field of computer crime has matured possessing rich history (Casey 2004; Maria 2011). The growth of the Internet has resulted in the increasing computer attacks and intrusions. Among these attacks, Malware (**mal**icious soft**ware**) is one of the biggest threats facing the digital world (Alperovitch 2011; RSA 2011). With more and more use of computers, portable devices and the Internet in everyday life, identification of new or unknown malware has become the biggest challenge in digital forensics (Vassil 2009). Of late, malware are being designed more for financial gain leading to a huge impact against individuals, organisations and business assets. Recent trends in malware designed for such financial fraud purposes indicate their increasing complexity and they are evolving rapidly as Internet provides more opportunities for automated financial activities. As a result, the financial damages caused by malware to individuals and businesses have dramatically increased in the past few years (RSA 2011).

In many ways, cybercrime is no different than traditional crime (Jahankhani & Al-Nemrat 2008). Both crimes involve in identifying targets, using surveillance and psychological profiling. The major difference is that the perpetrators of cybercrime are increasingly remote to the scene of the crime (Jahankhani & Al-Nemrat 2010). The traditional idea of a 'criminal gang' loses its meaning as members can now reside on different continents without ever having to actually meet.

Data has become more valuable than money. Hence, accessing bank data gives cyber criminals repeated access to the money. Researches into credit card fraud detection have steadily increased over the recent years (Aycock 2006; Ghosh & Turrini 2010; James 2007; Komisarczuk 2010). Intruders, who achieve unauthorized access to financial system for financial gain, cause losses for financial sectors and there is no single technique that can deter them. However, a threat that was once focused on single criminals is now focused on major organised crime crossing international boundaries and jurisdictions. Recent white paper (RSA 2011) to review the current state of cybercrime by the RSA Anti-Fraud Command Center warns that attacks will become more prevalent as more persistent techniques are adopted based on what they witnessed in the year of 2010 and 2011. Moreover, use of botnets, VOIP and mobile SMS in attacks are expected to rise. Globally, 30,000 phishing attacks are reported each month and at least three percent of phishing attempts are successful (RSA 2011). Although phishing alone is not directly responsible for all online banking fraud, Singh's statistical work in cyber security indicates that 900 online bank accounts get compromised each month from phishing alone (Singh 2007). Therefore, online banking fraud, in our case, includes all

unauthorized transactions conducted without the legitimate account holder's knowledge and (usually) resulting in loss of funds from the account.

Malware detection usually occurs in online systems and the anti-virus software forms the primary tool for the defense against malware. Though the quality of such malware detectors is improving in the techniques being adopted, namely 'virus signature' based detection or heuristic based detection, the malware attackers are always one step ahead of the anti-virus tools (Stolfo *et al.* 2007). The present malware detection systems usually rely on existing malware signatures with limited heuristics and are unable to detect those malware that can hide themselves during the scanning process in online systems (Alazab, Venkatraman, *et al.* 2009; Venkatraman 2009). According to a recent report (RSA 2011) new malware goes undetected by the commercially available anti-virus tools, and the literature survey conducted in this study indicates the need for new techniques to identify such hidden malware. Therefore, in order to address this requirement, this research study concentrates, first to developing a robust digital forensic process for NTFS file system, and then to design and apply innovative techniques for fulfilling the main objective of detecting hidden malware, which was still an unsolved challenge for malware detectors. Thus, the research project presented in this document attempts to fill the gap in both literature and practice.

As a first step to address the challenge, Chapter 3 investigates offline NTFS file systems and propose effective digital forensic techniques that could be used to analyse and acquire evidences of hidden malware in NTFS disk images. In the same chapter, the research focuses on achieving the main goal of this research in proposing and evaluating an innovative methodology to effectively detect hidden malware. Since NTFS is

predominantly used in most computer systems, and malware attackers take advantage of its weaknesses to hide malware, this research investigates and identifies the main areas of hidden malware growth. With this in view, the research aims i) to explore the NTFS disk structure and its vulnerabilities (Alazab, Venkataraman, *et al.* 2009), ii) to investigate weaknesses of existing commonly used digital forensic techniques (Alazab 2009) such as signature-based (Preda *et al.* 2008), anomaly-based (Patcha & Park 2007), and iii) to propose and evaluate improved methods in static analysis of NTFS for identifying hidden malware by investigating the disk image (physical), and for detecting unknown malware through file content (logical) analysis. Preliminary investigations conducted in this research study to identify hidden malware using function call analysis have been reported (Alazab, Layton, *et al.* 2010; Alazab, Venkataraman, *et al.* 2010) and subsequently using file content analysis.

Sophistication in malware through code obfuscation has created another challenge for digital forensic examiners and reverse engineering, namely the detection rate of new and unknown malware is low rate (Passerini *et al.* 2009; Stang 2010) and identifying benign code as malicious, which is termed as false alarm rate, is high (Patcha & Park 2007). The second challenge is extracting features from the obfuscated executables for reverse obfuscation is labor intensive and requires deep understanding of kernel and assembly programming. This thesis aim to develops fully automated system to extract two independent features, namely OP code and API function call features for finding the fingerprint of executable programs and for detection and differentiation of different files that are either malicious or benign. Obfuscation techniques aim to reduce vulnerability to any kind of static analysis for the determent of any reverse engineering process.

Extracting the payload, the de-obfuscated malware, and maliciousness hidden is a challenging task. This thesis aims to provide solutions to the automatic extracting of features out of binaries.

The Third challenge is unknown or obfuscated malware detection. Literature studies on malware detection as illustrated in Chapter 2 have shown that there is no single technique that could detect all types of malware (Chouchane & Lakhotia 2006; Dinaburg *et al.* 2008; Lawton 2002; Sharif *et al.* 2008; TreadwellZhou & Zhou 2009). Two techniques are commonly used for malware detection: signature-based detection and anomaly-based detection. Anti-malware engines use malware signatures to detect known malware. However, these countermeasures cannot detect unknown malware or unknown signatures which uniquely identify a specific malware. Therefore, signature based approaches fail to detect unknown malware. On the other hand, anomaly-based detection uses the knowledge of normal behaviour patterns to decide the maliciousness of a program code. It has the key advantage and ability to detect zero day attacks. However, it is very difficult to accurately specify the system or program's behaviour and thus these approaches usually result in a high false positive rate. For the second challenge in this dissertation; signature-free detection methods are proposed in Chapter 4, Chapter 5 and Chapter 6 to cope with polymorphic transformations and metamorphic obfuscations of malware, and use supervised machine learning algorithms for building better anomaly detection:

1) OP-code Features based Malware Detection: detailed in Chapter 4, using the knowledge of normal behaviour patterns of the x86 IA-32 operation codes (op-codes), this research work has proposed a novel algorithm that combines op-code frequency statistics

and hybrid wrapper-filter based feature selection technique for constructing a classifier for malware detection. Also, hybridized op-code statistics with novel wrapper-filter based feature selection technique to optimise the process has resulted in achieving the desired efficiency for large datasets.

2) API Features based Malware Detection: detailed in Chapter 5, using the knowledge of normal behaviour patterns of the Application Programming Interface (API) function calls. While some research has been conducted in arriving at file birthmarks using API call features and the like, there is a scarcity of work that analyses deeply with respect to the use of such features in malcodes. To address this gap, an attempt for the first time has been made to automatically classify the behavior of the API function calls based on the malicious intent present in any packed program. This approach also provides scope for deeper understanding of code obfuscation and to reverse it automatically with least human effort as explained in Chapter 5. Also, Chapter 5 proposes a five-step methodology for developing a fully automated system to arrive at six main categories of suspicious behavior of API call features to optimise the process and to achieve the desired efficiency for large datasets by using support vector machine algorithm with n-gram statistical analysis of API calls by varying n-values from 1 to 5. The main aim is to increase the true positive rate, reduce the false alarm rate, and to improve the overall accuracy.

3) Malware Detection and similarity detection based on Data Mining of API calls, as detailed in Chapter 6, a machine learning framework is proposed and evaluated with large datasets to investigate further on the preliminary observed patterns and to analyse using a variety of data mining techniques for detecting malware from benign files

effectively, based on the frequency of occurrence of each Windows Application Programming Interface (API) calls found in the datasets. Also, different distance measures have been implemented and similarity analysis performed by using eight commonly used distance measures in vector models, namely Cosine, Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Euclidean, and Hamming distance similarity measure for Nearest Neighbor (NN). As well as this, a supervised learning approach has been adopted that uses a dataset to train, validate and test, an array of classifiers. Robust classifiers have been selected, namely Naive Bayes (NB) Algorithm, k−Nearest Neighbor (kNN) Algorithm, Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels (SMO - Normalized PolyKernel, SMO – PolyKernel, SMO – Puk, and SMO- Radial Basis Function (RBF)), Backpropagation Neural Networks Algorithm, Logistic Regression, and J48 decision tree. The chapter also provides details of how data will be collected to conduct the experimental analysis and the data mining algorithms adopted for the study will be evaluated.

In creating new malware, malware authors use obfuscation techniques and behavior modification in order to thwart malware detectors. Obfuscation attempts to hide the true intentions of existing malicious code without changing the behaviours exhibited by the malware. Behaviour modification creates new malware by making changes to the existing malware, although the same behaviour of the malware stills the same. Sophisticated tools are available for malware writers to create new malware very quickly suiting their needs based on these techniques. Reuse and recycle code is a major component in the development of new malware effortlessly.

1.2 Information Security

Free from danger is the simplest definition for security, and information security is the protection of information from any kind of harm. Protection of information from malware authors (who are called under a variety of names such as black hats, hackers, and crackers) is of utmost importance in today's information society with high level of cybercrime. Currently the internet offers the biggest buyer and seller of goods and services through electronic medium. Hence, as today's society becomes more electronic in terms of smart cards, electronic money, electronic purse, electronic checks, digital cash, stored value cards and online banking, more opportunities for serious threats to e-security have been created. This threat evolution has escalated to a great extent as society has moved towards online as the preferred method for Identity theft, financial transactions and payments (Turville *et al.* 2010). Today's information security is being breached by such threats posed to individuals and organizations as cybercrimes continue to aggressively develop techniques to steal money and financial credentials or personal information for financial gains.

1.3 Computer Forensic

More than quarter century ago, in 1984, the FBI Laboratory and many other law agencies had commenced developing structures to examine computer evidence, in order to address the growing demands of computer crimes (Vacca 2005), and the FBI had established the Computer Analysis and Response Team (CART). In 1991 the term 'Computer Forensics' was coined by the International Association of Computer Investigation Specialists (IACIS) in Portland, Oregon (Vacca 2005). *Computer forensics* is the science of

preserving, identifying, extracting, analysing and documenting computer evidence found at crime scenes so that this evidence may be used in a court of law (Rogers & Seigfried 2004). It also answers questions and attempts to provide full descriptions of a digital crime scene (Reith *et al.* 2002). In computer systems, the primary goals of digital forensic analysis are fivefold: i) to identify all the unwanted events that took place, ii) to ascertain their effect on the system, iii) to acquire the necessary evidence to confirm malicious activity that may have occurred on computer systems, iv) to prevent future incidents by detecting the malicious techniques used, and v) to recognize the incitement reasons and intendance of the attacker for future predictions. The focus of this research is on goal iii) as this goal has become a major challenge with the recent increase in hidden malware whose malicious activities go unnoticed by current detection tools and techniques.

Computer forensics is a new science that deals with both law and electronic devices (Reed 1990-91). The number of criminal justice agencies and organisations is being constantly increased and they share responsibility for detecting and stopping digital crime. The Regional Computer Forensics Laboratory of the Federal Bureau of Investigation (FBI), in their annual report (RCFL 2008) stated that 1,756 TBs of data was processed in 2008 alone. The year before (RCFL 2007) the RCFL announced that the amount of data examined per criminal case was increasing by 35% annually from 83 Giga bytes in 2003 to 277 Giga bytes in 2007.

Digital electronic evidence can be described as the information and data of investigative value that are stored by an electronic device (Casey 2004; Kruse & Heiser 2001). There is no single universal procedure to conduct the investigation of a digital crime scene (Carrier 2005). Furthermore, there is no standard process, framework or

model for conducting forensic investigation in Windows NT File Systems (NTFS), which is the prime motivation of this research. Many situations have adopted three major phases for investigative process, which are; acquisition, preservation, and analysis (Andrew 2007). However, recent trends of hidden malware warrant new techniques for digital forensics. Hence, this research work aims to investigate and improve the digital forensic techniques that could be used to analyse and acquire evidences of cybercrimes and attacks from the most commonly used file system on computers, namely, Windows NT File System (NTFS).

Digital investigation is a process to answer questions about the compromised digital data and this involves using either static or live analysis techniques (Kruse & Heiser 2001). Even though live analysis techniques could help in capturing evidence during forensic investigations to a certain extent, they are far from infallible and lead to false negatives of hidden malware. In live analysis, malware such as rootkits can hide and change itself without being seen. Moreover the attackers target a hidden area on the system structure to hide the malware. Hence, this research focuses on static analysis technique as all the hidden information can be captured and cannot be modified to produce false data as in case of live analysis techniques. An image copy of the NTFS hard disk would capture even hidden data and hence this study entails developing efficient techniques to analyse the data in a confined lab environment for the identification of hidden malware. This research work formed a major initial step towards addressing the open problem of identifying unseen or new malware that could evade detection in the form of hidden or obfuscated malicious code.

Forensic investigation was conducted on NTFS with existing forensic tools by running the disk image in a trusted operating system to search for evidence, as shown in Chapter 3. It was observed that since the NTFS disk image records every event in the system, the data required to be analysed is huge and this has led to imperfect forensic tools that are practical for real-time implementation but not comprehensive and effective as they were unable to detect all malware, in particular hidden malware. Therefore, this preliminary investigation confirms that a comprehensive methodology with an improved technique is warranted for an effective forensic analysis that is capable of detecting hidden malware.

1.4 Definitions

Malware: has numerous synonyms such as; malicious software, malicious code (MC) and malcode. Malware contains code designed to perform illegal activities that cause damage and affect the integrity and functionality of digital electronic devices. McGraw and Morrisett (McGraw & Morrisett 2000) define such malicious code as "any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system". For the purpose of this research, the description given by (Vasudevan & Yerraballi 2006) has been adopted for malware as a generic term that encompasses viruses, worms, Trojans, exploits, backdoors, keystroke loggers, rootkits, spyware, and spam. These terms are coined based on the functionality and behaviour of the malware.

Viruses: The father of computer viruses Dr. Frederick Cohen defined the term computer virus as 'A virus is a program that is able to infect other programs by

modifying them to include a possibly evolved copy of itself' (Cohen 1987). The term computer virus comes from the similarities with biological virus since it infects a healthy subject and destroys it. In order for the virus to function and cause damage, it needs an existing host program. For instance, a virus has been reported and noted to modify the program code to take control of operating system, make copies of themselves to spread to new targets, and usually it attaches itself to commonly used software such as Adobe Acrobat, spread sheets or word processors. The most important part to note here is the new copies of the virus do not have the exact clone of the initial instance, this called 'metamorphic' viruses, sample functionality of the main one but with different byte sequence or the signature for each copy to be completely different (Szor 2005).

Worms: Computer worms have grown to become the fastest spreading and most costly malicious code threats (Holz *et al.* 2006). A worm replicates itself by executing its own code independently, and spreads to the other computers using network resource connections such as e-mail, TCP/IP, IRC, etc. with the goal of infecting as many computer systems connected to the network as possible. Another name for computer worms given is network viruses. Since computer worms are designed to copy itself from one computer to another, while viruses are designed to spread themselves from one file to another on a single computer, worms are more complex than viruses. The other primary distinction between a virus and a worm is that a worm does not need a host to cause harm unlike a virus.

Trojan horses: A Trojan horse is a seemingly harmless computer program designed to get access to the computer from another location and to perform unauthorized action by tricking computer users to run program masquerades as a legitimate program.

For example, the program could contain a useful function (such as local weather) to entice users to run the program. Trojan horses replicate in a different way than the replication procedure adopted by viruses and worms. However, a Trojan horse can be part of the payload of a worm and can be spread to many machines as part of a worm infection. It has also been reported that Trojan horses mostly have been sent out as email attachments (Szor 2005). Usually, Trojan horses are associated with accessing and sending unauthorized information from their host via email which could be classified as spyware as well. The embedded malware could also be a time bomb designed to activate at a certain time (Landwehr *et al.* 1994).

Logic Bomb: created by malicious authors through inserting the malicious code into a system which remains dormant and executes its payload when malicious function conditions are met (such as pre-defined time), and is also named 'slag code'. However, a logic bomb can be used by a virus, worm and other malicious code to gain power, and spreads before being noticed. When it gets executed, it could cause Denial of Service (DoS) that leads to slowing down of system or a storage overflow, and would remain dormant until activated.

Rootkit: Rootkit name comes from the combination of two words, "root" and "kit". The highest access level in UNIX environments is "Root", while "kit" refers to tools. Hence, a rootkit is defined as (Manap 2001) a collection of tools that enable attacker to have full control by obtaining the root access level on the compromised system in order to hide its payload from the operating system without being detect. A rootkit can hides its presence by hiding the actual files, processes, network connections, the sniffers, etc. so that there may be a number of processes running on a system that are

not revealed in Task Manager or established connections or **netstat** display. The rootkit is able to do this by manipulating function calls to the operating system and filtering out information that would normally appear. Once a rootkit is installed, it allows an attacker to mask the ongoing intrusion and maintain privileged access to the hacker by circumventing normal authentication and authorization mechanisms (Lobo, Watters, *et al.* 2010a, 2010b).

Botnet: is a collection of computers which interact together to compromised computers and to accomplish their illegal goals. The compromised computers are referred to drones or zombies, and the malicious software running on them as 'bots'. Botnets send the majority of spam (McCombie *et al.* 2009). Bots use vulnerable machines with methods which are used by other malware classes and they use the command and control (C&C) channel.

Spyware: is a software that is sneakily installed on a computer to collect information on system without the user knowledge or consent. However, this software can be classified as a Trojan horse as well. The consequences of spyware infections can be severe, including inundating the victim with pop-up ads, stealing the victim's financial information or passwords, or rendering the victim's computer useless (Moshchuk *et al.* 2006).

Backdoor: is a software program which is installed by the attacker on a compromised system to bypass normal security controls on a system and facilitate the further unauthorized access of the attacker on the system.

Exploit: is software, a chunk of data or a sequence of commands that makes use of the vulnerabilities of the victim computer to create an unwanted action on the victim's computer. These actions include obtaining the control of the computer system, access control destruction, or DoS. Often exploits are used to install viruses, warms and rootkits.

1.5 Infection Strategies of Malware Authors

As shown in the previous Section (1.4) there are many types of malware, also different instances of malware have a variety of penetration methods, malicious purposes, and effects. A malware instance can be transported through remote exploit, by an e-mail, over a peer-to-peer network, or through removable media. It can also be automatically downloaded and installed by visiting a web site containing exploit code (Moshchuk *et al.* 2006).

The ability to execute program tasks without interrupting the user is very common in programming. A program will not want to interact with a user when an error occurs for reasons beyond the user's comprehension. Malware authors implement the same technique to prevent a user from suspecting infection. This can be very dangerous, as a virus is allowed to execute fully, without the expressed permission of the user, or indirect feedback from the computer.

Criminals today have sophisticated service providers and high-tech expertise to fully take advantage of their current targets. Furthermore, the exploit servers used can be changed to avoid detection and countermeasures. Malware authors use variety of methods in order to avoid detection, and employ different kinds of deception (Watters &

McCombie 2011). The most common strategies adopted are summarised in the following subsections (Aycock 2006; Skoudis & Zeltser 2003):

1.5.1 Overwriting Infection

An overwriting infection is accomplished by inserting malicious code into an area of the original file of the host computer such as XLS, DOC or PDF. Some type of malware using this strategy completely overwrites a file which destroys the original file, rendering an entire program useless. However, the advanced malware do not try to destroy the file in order to trick the user and evade detection tools. This is done by adding the malicious code in the head or the tail of the host file in order to not affect the functionality of the host file (Crescenzo & Vakil 2006).

1.5.2 Companion Infection

A companion infection is different from overwritten malware as it does not require a host file to insert the malicious code; instead it creates a companion file to the EXE host file (Jacob *et al.* 2009). To test this infection, a simple experiment performed by injecting the windows XP professional system with a malware and named it similar to the original windows file ***calc*** but with the extension COM (calc.COM) and saved it in the same path of the original file of calc.EXE as shown in Figure 1.1. Many users have a tendency to type the name of the file at the 'run' command without typing the extension in order to launch an executable program. Since Microsoft DOS looks for .COM files to be executed before it looks for .EXE files, malware authors need only to save or copy itself as .COM in the same directory as the original .EXE launched. Malware authors are taking this

advantage and they not only save the file with .COM at the same path as the .EXE, but they also conceal their existence by assigning a "hidden" attribute to the companion malicious file (the COM file). This way they try to decrease the likelihood that the system's user could discover the companion file in the directory listing as by default, files with this hidden attribute do not appear in directory listings. Alternatively, the attacker tricks computer user into executing malicious code by creating a malware file with the same name as the benign program, and placing the malicious executable earlier in the path than the benign one (Skoudis & Zeltser 2003).

Figure 1.1 Companion Infection example

1.5.3 Appending Infection

Also called hooking infection an appending infection is a method is predominantly adopted by the rootkits as they implant various hooks such as Import Address Table (IAT) hooks, inline function hooks, System Service Descriptor Table SSDT hooks, etc. appending infection uses operand command instructions (such as jump (JMP) and CALL), by modifying the first few instructions in the file function so that the execution jumps to the address pointed to the malicious function. Usually the malicious code is inserted at the end of the file where it is not affected by the functionality of the file and does not raise any suspicion. An example of such a rootkit is the virus ‘Vienna’. The

infection technique can be used in all type of portable executables (PE), such as EXE, NE, ELF, etc (Lobo, Watters, Wu, *et al.* 2010).

1.5.4 Prepending Infection

Prepending infection is similar to appending infection, except that the malware inserts itself at the start of a file. Malware authors have implemented this type of infection on various operating systems (Daoud *et al.* 2008). Since the malicious code is allocated at the start of the file, the malicious prepending code runs before the original code. An example of the use of this technique is the Hungarian virus Polimer.512.A, which inserts itself within the first 512 bytes of the beginning of the executable and shifts the original program content to follow this block of code.

1.5.5 Cavity or space fill infection

Cavity or space fill infection is an infection that attempts to enclose the malicious code in an empty space while not affecting the actual program and at the same time, not increasing the length of the program. Malware authors infect files without increasing their size or damaging the files. They accomplish this by overwriting unused areas of executable files. These are called cavity viruses. For example, the CIH virus, or chernoby1 virus uses this strategy to infect portable executable files. Also, in the Microsoft NTFS file system, Master File Table (MFT) is the core of NTFS since it contains details of every file and folder on the volume. Each MFT entry has a fixed size which is 1 KB containing two attributes; an attribute header and attribute content. The attribute header is used to identify the size, name and the flag value. The attribute content

can reside in the MFT followed by the attribute header if the size is less than 700 bytes (known as a resident attribute), otherwise it will store the attribute content in an external cluster called cluster run (known as a non-resident attribute). This is because the MFT entry is 1KB in size and hence cannot fit as it occupies more than 700 bytes. The Lehigh virus is an example of this infection (Alazab, Venkataraman, *et al.* 2009).

1.5.6 Boot Sector Malware

Boot sector malware infection involves infecting the boot sector in order to infect the system every time the machine boots up. The boot sector placed in the beginning of each partition is appropriately called the partition boot sector (PBS) and the Master Boot Record (MBR) is allocated in the first sector of the hard drive which contains the boot code (Lobo, Watters, Wu, *et al.* 2010). Hence, when the system starts, it locates the first sector on the hard drive, and executes MBR. Boot sector malware infect the Master Boot Sector of the hard drive and infect the system every time it boots up. The Michelangelo virus is an example of a Boot sector malware.

1.5.7 Macro Malware

Embedded in a data file, the term macro means a series of command steps and character strings saved in a single location that is assigned a name (Alsagoff 2011). Malware authors are using macros to infect applications such as word processing and spreadsheet files. Usually, when the name of the macro is found in a document file, the macro is expanded. The macro could perform the series of steps automatically for manipulating

and creating files, changing menu settings, etc (Yao & Liu 2011). A macro virus is often spread as an e-mail virus. A well-known example is the Melissa virus in March, 1999.

1.6 Malware on the horizon

In this research the focus is to conduct static analysis effectively because all the hidden information can be captured and cannot be modified to produce false data, which is apparently the main drawback in live analysis techniques (Alazab 2010). Dynamic analysis techniques analyze the code of a program by actually executing it. They are robust to the obfuscation techniques and those anti-static-analysis techniques (i.e, self-modifying). However, dynamic analysis has many drawbacks. First, it incurs much more overhead than the static analysis. There could be a lengthy code sequence that has to be executed to reach conclusions about the code behaviour. Second, it only covers a part of all possible program execution paths. Therefore, many important behaviours of the analyzed program cannot be discovered. Third, it is hard to simulate the execution conditions under which the analyzed malware exhibits its malicious behaviour. For instance, a bot program needs to receive control and command from a bot master to exhibit its malicious behaviour. Fourth, if the code is executed in a virtual machine, there are techniques that can be utilized by the attackers to determine whether the code is running in virtual environment. As a result, the code can be designed to modify its runtime behaviour.

The research project presented in this document concentrates on detecting hidden malware, which is still an unsolved challenge for malware detectors (Stolfo *et al.* 2007). As a first step to address this challenge, this research would investigate offline file

systems and improve the digital forensic techniques that could be used to analyse and acquire evidences of hidden malware in NTFS disk images.

The main goal of this research is to propose a methodology to effectively detect hidden malware. Since NTFS is predominantly used in most computer systems, and malware attackers take advantage of their weaknesses to hide malware, this research focuses on main areas of hidden malware growth in NTFS based systems. With this in view, the research aims i) to explore the NTFS disk structure and its vulnerabilities, ii) to investigate weaknesses of existing commonly used digital forensic techniques such as signature-based, heuristic-based and anomaly-based, and iii) to propose and evaluate improved methods in static analysis of NTFS for identifying hidden malware by investigating the disk image (physical) and by detecting unknown malware through file content (logical) analysis.

Since malware detection techniques work very well in detecting known malware, the malware authors have come up with new and improved techniques for their code to hide and evade detection by using many techniques, such as polymorphic and metamorphic techniques. This has led the AV vendors to start studying the behavioural analysis of the file to check whether the file is benign or malware.

Techniques used for malware detection can be classified into two categories: anomaly-based detection and signature-based detection. Since the major drawback for signature-based detection is the inability to detect new or unknown malicious code and zero day attack, the focus of this research is on anomaly-based detection, a technique that uses the knowledge of what is under consideration to find out what actually is malicious.

In anomaly-based detection, the inverse knowledge comes from the learning phase, Chapter 4 the knowledge was based on operand codes, Chapter 5 and Chapter 6 the knowledge was based on API function calls. So, anomaly-based detection alerts what is anomalous behaviour based on knowledge of the normal files. Malware detectors usually take two inputs. One input is the knowledge of the malicious behaviour, which comes from the learning phase. The other input is the file under inspection or testing that is analysed to decide if the file is malicious or benign.

1.7 Research Questions and Hypotheses

Malware identification and analysis is a technically intense topic, requiring deep knowledge of multiple computer science disciplines. To compound the problem, successful identification and analysis by malware analysts has been confounded by the use of hidden and obfuscated malicious binaries in recent years. Cybercriminals have adopted various obfuscation techniques to disguise the malware binaries, making it difficult to identify. There are available open-source and commercial tools which are being used to make the malicious code highly obfuscated in order to remain hidden in the NTFS file system hard disk drive without being detected by anti-virus tools.

The majority of anti-virus detection systems are signature-based detection, which is the method used by most of anti-virus detection engines as it is highly effective in detecting known malware. Even though hybrid systems applying heuristics have been recently explored, today's anti-malware approaches are neither efficient nor effective in detecting current obfuscated malware attacks. Therefore, this research aims at proposing

and presenting effective and efficient techniques for detecting hidden and obfuscated malware.

With the recent trend in malware to operate as obfuscated malicious code to remain hidden and evade from detection any live analysis technique or anti-virus tools, this research attempts to identify such unknown malware that adopt the following two predominant methods to attack:

- Physical space such as hidden malware in boot record or slack space, and
- Logical space such as code obfuscation of binary contents.

The primary purpose of this research is to answer the following research questions:

Q1: Could malicious code hidden in NTFS file system physical space be detected using an automated process?

Our hypothesis is that signature based detection and existing forensic analysis tools are unable to reveal hidden malware in NTFS physical space and it calls for a new forensic analysis process.

Q2: Could anomaly based detection using static features be applied to effectively detect and classify obfuscated malicious code attacking through binaries or executable files launched in the logical space of computer systems?

Our hypothesis is that existing live analysis malware detection techniques and anti-virus tools that predominantly use signature based methods are unable to detect obfuscated malware hidden in binaries of the computer logical space, and hence calls for

new anomaly based detection techniques that could identify obfuscated malware features by conducting static analysis of the behaviour properties exhibited by such malware.

1.8 Research Methodology

To address the first research question (Q1) listed in section 1.7, this research requires a systematic methodology to search computer hard disk space to search for significant evidence of hidden malware attacks. We adopt the Integrated Digital Forensic Model (IDFM) by Carrier & Spafford (2003) that is widely reported in literature for digital forensics. We propose a new forensic analysis process within the IDFM framework to detect hidden malware in computer physical disk space such as boot record and slack space.

For the second research question (Q2) listed in section 1.7, this research adopts static analysis methodology as it is well suited for byte-level content analysis of malicious patterns that is not possible using live or dynamic analysis conducted in existing anti-malware tools (Karim *et al.* 2005; Kotler and Maloof, 2006). Since signature based detection and other existing methods are unable to detect and classify the hidden malicious activities, this research proposes novel methods of extracting anomalies in the behaviour patterns of obfuscated malware. We study their statistical behaviour properties by analysing patterns such as op-codes, API calls, n-gram byte sequences efficiently to identify and classify unknown malware hidden in the binaries of the computer logical space. A variety of data mining and machine learning techniques are adopted and the results are compared in order to determine the performance of our approach.

1.9 Contributions

In this dissertation, multiple research problems related to the infection strategies of malware authors (Section 1.5) and code obfuscation explained in (Section 2.7) have been studied. These techniques attempt to bypass the most popular malware detection method, signature based detection. The overall observation is that malware authors are producing unique threats using different obfuscation methods, and signature-based detection is of little defense to our present computing environments and such traditional anti-virus techniques are rapidly becoming obsolete. Therefore, Anomaly Detection (AD) should be explored and used rather than signature-based detection. Also, anomaly-based detection methods are required to be adopted to detect malicious activities that are increasing exponentially since the start of this year.

The literature review presented in Chapter two has clearly identified the lack of existing digital forensic methods and techniques for identifying hidden malware, which could be either in the form of slack space implantation (physical) or as obfuscated code (logical) in file content within the NTFS file system.

The first contribution will fill the gap in literature and practice as there is no standard process, framework or model for conducting a comprehensive forensic investigation in NTFS slack space (Purcell & Lang 2008). By effectively extract features of slack space for detecting hidden malware of the first form will be given. Since malware attackers take advantage of NTFS file system weaknesses and the inability of existing AV to check the slack space, a guideline to investigate slack space for identifying known and unknown malware forms a significant contribution of the study.

Chapter 3 of thesis provides knowledge, methodology and discusses the analysis techniques used to successfully detect maliciousness in hidden data and hidden space, by investigating the NTFS file system boot sector.

Sophistication in malware through code obfuscation has created another challenge for digital forensic examiners and reverse engineering, namely the detection rate of new and unknown malware is low rate (Passerini *et al.* 2009; Stang 2010) and identifying benign code as malicious, which is termed as false alarm rate, is high (Patcha & Park 2007). Extracting features from the obfuscated executables for reverse obfuscation is labor intensive and requires deep understanding of kernel and assembly programming. Chapter 4 and 5 in the thesis provide methodologies, develops fully automated system to extract two independent features, namely OP code and API function call features for finding the fingerprint of executable programs and for detection and differentiation of different files that are either malicious or benign.

A signature-free detection method is proposed to cope with packer, polymorphic transformations and metamorphic obfuscations of malware, and use knowledge parts for building better anomaly detection. For effective and efficient solutions, this thesis moves away from the signature based detection to anomaly based detection. This thesis provides two solutions to the limitation of signature based detection. First, the detection of malware uses the knowledge of normal behaviour patterns of the x86 IA-32 operation codes (op-codes) and a novel algorithm is proposed that combines op-code frequency statistics and hybrid wrapper-filter based feature selection technique for constructing a classifier for malware detection, as shown in Chapter 4. Also, hybridized op-code statistics with novel wrapper-filter based feature selection technique to optimise the process and

achieve the desired efficiency for large datasets. Experimental results show that our frequency-statistics based approach achieves high accuracy ~96%. Second, detection Malware based uses the knowledge of normal behaviour patterns of the Application Programming Interface (API) and proposes a five-step methodology for developing a fully automated system, also, investigates patterns of obfuscated code further using several data mining techniques aiming to increase the true positive rate, reduce the false alarm rate. Chapter five has used statistical n-gram analysis, feature extraction, feature selection, using SVM algorithm of binary content based on system call sequences together with innovative techniques to classify whether the binary content is benign or malicious would improve anomaly-based detection. Chapter Six, used the automated data mining system implemented for this study has achieved high true positive (TP) rate of more than 98.5%, and low false positive (FP) rate of less than 2%, which has not been achieved in literature so far. This is much higher than the required commercial acceptance level indicating that our novel technique is a major leap forward in detecting zero-day malware.

This thesis also proposed similarity based detection of unknown malware and obfuscated malware using API function calls features, using various distance measures of vector models. As shown in Chapter 6 results show that our proposed method is an effective method to accurately differentiate malware from benign files and, more importantly, to detect obfuscated malware families.

1.10 Roadmap of the Dissertation

The rest of the dissertation is organized as follows. In the next chapter, the background of study and related work is provided. A literature review of related works is also presented in Chapter 2 along with the research contribution and significance of the study. Chapter 3 discusses the forensic analysis of the NTFS file system and existing problems surrounding forensic analysis, such the vulnerabilities identified in the NTFS file system disk structure and the weaknesses present in the current forensic techniques. These form the main motivation in proposing the research questions that have been investigated in this project with the aim to address hidden malware problem in both physical and logical content of computer systems. Also, the proposed forensic analysis techniques could be used to detect hidden malware effectively by analysing the internal structure of the NTFS disk image (physical). Anomaly based detection uses the knowledge of normal behaviour patterns of the x86 IA-32 operation codes and the proposed novel algorithm that combines op-code frequency statistics and hybrid wrapper-filter based feature selection technique for constructing a classifier for malware detection is described in Chapter 4. Chapter 5 presents an automated method of extracting API call features and analysing them in order to understand their use for malicious purposes. In addition, a five-step methodology is proposed for developing a fully automated system to arrive at six main categories of suspicious behaviour of API call features. The methodology is devised to detect obfuscated malware by investigating the structural and behavioural features of API calls. In particular, n-gram statistical analysis of API calls is applied and experimental results with large datasets have been analysed for performance and accuracy. Chapter 6 has two detection methods proposed. First, based on similarity detection, this research proposes a

new method to identify zero-day malware that is obfuscated from an existing malware family by employing similarity measures for Nearest Neighbor (NN) search of Windows Application Programming Interface (API) call features. The second detection method is anomaly detection, where a data mining framework has been proposed to detect zero-day malware effectively as it learns through analysing the behaviour of existing malicious and benign codes in large datasets. Finally, Chapter 7 provides the conclusions of this study, highlighting the contributions of the research work and recommendations for future work.

Chapter 2 : Background of Study and Literature Review

"Is there any point to which you would wish to draw my attention?"

"The dog did nothing in the night-time."

"That was the curious incident," remarked Sherlock Holmes.

— Sir Arthur Conan Doyle, *"Silver Blaze,"*

The Strand Magazine (1892)

2.1 Introduction

Currently, the internet is the biggest platform where buyers and sellers of goods and services transact through an electronic medium. Today's society is getting more dependent on electronic technologies such as smart cards, electronic money, electronic purse, electronic checks, digital cash, stored value cards and online banking. This has created opportunities for serious threats to e-security. Cybercriminals facing current threats to organizations continue to aggressively hunt and develop new techniques to steal money and credential information, thereby resulting in an exponential rise in cybercrime year after year (RSA 2011). In the context of crime-ware, malware is the most valuable resource to perform unauthorized access by cybercriminals (Ghosh & Turrini 2010).

Cybercriminals are using a variety of highly sophisticated techniques to fool, thwart and evade any commercially available detection engine. Therefore, 'Free from danger' has become hard to achieve and is a dream for daily internet users. Malware

affects the secrecy and integrity of data as well as the control flow and functionality of a computer system. Escalating increase in commercial and financial transactions conducted online provides opportunities for cyber criminals to conduct unauthorized access to digital systems. Criminals use zero day vulnerabilities to conduct their activities and anti-forensics techniques to evade being tracked (Alperovitch 2011).

Indeed, a review of the history of malware (Ghosh & Turrini 2010; Venkatraman 2009) and anti-malware reports (Symantec Enterprise Security 2010, 2011a, 2011b) and predictions (Konstantinou & Wolthusen 2008) show a continuous cybercrime growth thriven in sophistication over the years, and traditional malware detections appear insufficient to tackle increasingly sophisticated malware. Therefore, the detection of malware is not only of interest to researchers but is also a major concern to the general public. Recent trends in malware for such malicious and illegal purposes indicate increasing complexity and are evolving rapidly as systems provide more opportunities for more automated activities of late. The damages caused by malware to individuals and businesses have dramatically increased recently. Hence, this forms the motivation for the research work to focus on the obfuscated techniques used in the malware in order to understand their behaviour and find patterns that would aid in detecting unknown malware. Since malware exploits and uses file system vulnerabilities to infect the systems, this research start by the growth and attack strategies of malware through an illustration of Zeus botnets, and identifies the various existing malware detection methods that fail to combat them. Hence, there is an imminent need for developing new malware detection techniques for addressing this situation of rapid malware evolution.

2.2 Growth in Malware

As the internet plays an essential role in all areas of society, it has major impact on the economy, with even the military and government functions of a country being targeted by malware. The Internet now connects billions of computers and has become an easy platform to implant malware attacks by exploiting vulnerabilities or flaws in software systems and critical applications so as to intentionally disrupt their use, or to subvert them for specific purposes. The rapid increase in malware threatens not only individual computers, but the availability of the Internet itself as systems are being taken control by the malware without the knowledge of the computer users.

Today the goals of those creating and unleashing malware have shifted, from simple vandalism and craving for recognition, to financial gain (James 2007; Stolfo *et al.* 2007). Malicious attacks have become more organized and purposefully directed. Botnets in particular confirm this trend (Khan *et al.* 2010). Botnets are armies of remotely-controlled computers, or zombies. These computers are compromised and then infected with software robots, or bots, that allow the zombie computers to be controlled remotely through established command and control channels (C&C). Collectively, under the control of C&C servers, botnets become powerful and effective slave computing assets that can be rented for illegal activities. Such activities include phishing attacks, installing backdoors or rootkits on host systems to obtain private information, sending spam for advertising, and launching large scale distributed denial-of-service (DDoS) attacks (Seewald & Gansterer 2010).

A recent major malware threat, the Zeus Trojan, a financial malware Zeus botnet, is a well-known banking Trojan also called Zbot, NTOS, WSNPOEM, or PRG, and forms the king of financial malware 'in the wild', both in terms of infection size and effectiveness (Seewald & Gansterer 2010). Furthermore, to date it is the biggest and the most sophisticated threat to internet security and to most of the detection engines such as Symantec (Symantec Enterprise Security 2011b) and McAfee (Alperovitch 2011). The Zeus Trojan is estimated to be responsible for about 90% of banking fraud worldwide (Alperovitch 2011) and found guilty in 44% of the banking malware infections, with 3.6 million PCs infected in the US alone (Trusteer 2009). Symantec Corporation describes it as "Zeus, King of the Underground Crimeware Toolkits".

The Zeus Trojan software, with a friendly interface toolkit that is available in underground online forums for $1,500 – $20,000US, enabling cyber criminals to configure and create malicious software to affect user systems, allowing them to take control of a compromised computer, harming the data, logging keystrokes, and executing unauthorized transactions in online banking. The name Zeus has created a panic in the world of computers and security experts today. Reports and studies show that since last year Zeus has been found embroiled in more than half of the banking malware infections in the world (Bitdefender Antivirus Technology 2010).

The Zeus Trojan primarily carries a very light footprint and is designed to steal sensitive data stored on computers or transmitted through web browsers and protected storage (Alazab, Watters, *et al.* 2011). Once infected, the computer sends the stolen data to a bot command and control (C&C) server via encrypted HTTP POST requests, where the data is stored (Alazab, Watters, *et al.* 2011). Also, it allows cybercriminals to inject

content into a bank's web page as it is displayed in the infected computer browser in real time. It is setup such that the stolen data is sent to a "drop server" controlled by an attacker called a botmaster and it allows cybercriminals to control the infected systems remotely. Moreover, Zeus is highly dynamic and applies obfuscation methods such as polymorphic encryption and metamorphic in a network of bots. In each infection, it re-encrypts itself automatically to create a new signature to defeat signature-based detection that makes the signature difficult to comprehend (Alazab, Venkatraman, *et al.* 2011). However, The Windows Zeus is increasingly hard to combat as it can successfully evade commercial detection engines and is able to hide malicious features such as string and API function calls. Zeus trojan is still developing and it has versions and new plugin releases that can also infect latest operating systems such as Windows 7 and Vista.

As a fresh threat, according to numerous research labs and hacker forums, the Zeus botnet recently has combined with the new release of 2010 'SpyEye Trojan' source codes to create more sophisticated bots and takes the malware threat to a new level (Maria 2011). This new toolkit is being reported to be currently available for purchase in the underground market and version 1.4.1 has been published on January 11, 2011 (SPAMfighter News 2011). The new version of the combination has two versions of a control panel used for committing fraud and managing compromised systems. Three trends, including the growth of the Internet connectivity, system extensibility and complexity, contribute to the growth and evolution of this problem (RSA 2011). The mono-culture nature of current both hardware and software makes it fairly possible to exploit a vulnerability which will infect a large number of host computers. The increasing connectivity of computers via high-speed Internet connections increases the visibility of

vulnerable systems and exposes them to these attacks (Altunaya *et al.* 2011). These trends indicate that self-learning and self-updating by observing system anomalies and behaviour patterns is much warranted in malware detection systems of the future (Venkatraman 2009).

Popularity of the Internet is growing day by day and millions of users are connected to each other on a daily basis. At the same time malicious activity is also growing with increasingly profit-driven motives and sophisticated evasion techniques. High speed data services, cheap broadband, low-cost mobile computing, remote access using online banking, Instant Messaging (IM), Internet Relay Chat (IRC), Common Internet File System (CiFS), Simple Mail Transfer Protocol (SMTP) programs that support Hypertext Markup Language (HTML), scripting, Peer-to-Peer (P2P), as well as new operating systems with vulnerabilities, have all helped cybercriminals to propagate their malicious code faster and even helped them to create highly effective malware in a way those malware can thwart the detection engines.

Creating and producing malicious code is not done only by malware writers, but there are also, malware kit vendors (Komisarczuk 2010) such as Zeus, exploit kits, Flesta, MyPolySploit, Limbo2 and SpyEye. These kits are used to create highly effective malware, serving as new offspring of malware. The new market for malware creation software on-sale is widely available on the internet and can be found easily using Google and other search engines. Malware kit or 'crime ware' is being offered for sale on underground trading forums and IM for negotiation (Emigh 2006). Apart from purchasing these kits, one could also buy the updates for the kit thereby ensuring and guaranteeing it as a reliable ongoing business. Likewise, cybercriminals are being hired

in underground markets with even after sales services and offers of guaranteed effectiveness of evading security countermeasures (Danny 2010). As a result, cybercriminals update the construction kits to suit the needs of their client base to stay ahead of their contenders.

"Full ZeuS Souurce code of last v2.0.8.9 (includes everything). Requires MSVC++ 2010. You can create your own HWID licenses and much more".

According to the Internet Crime Complaint Center (IC3)[2], malware are evolving rapidly more recently. A study conducted by University of Maryland shows that on an average, a computer connected to the Internet may experience an attack every 39 seconds. Equally important in the first quarter of 2010, another experiment conducted by the San Diego Supercomputer Center (SDSC) shows that an average of 27,000 hacking attempts were made per day. Since 2010, these figures have grown exponentially (Ghosh & Turrini 2010; Komisarczuk 2010; RSA 2011).

Currently, known malware can be recognized by all of the popular AV engines. However, attackers continually develop new techniques for creating malware that cannot be detected by AV engines. Once new malware is released, the AV engines will eventually update their signatures to combat the new malware. The growing size of the signature databases illustrates the mounting threat of malware. In February 2006, BiDefender Antivirus (Technology 2006) published that it had over 270 thousand malware signatures in its database. In 2009, Symantec Internet Security Threat Report announced that malware activity continues to grow at a record pace, and there are over 1.5 million new malware instances, mostly developed in 2008 (Symantec Enterprise

Security 2009a). The increasing size of malware signature databases forces AV engine researchers and developers to think about more effective methods to check the signatures rather than traditional signature based techniques. Another reason for the growing sizes of the signature databases is that new malware propagation mechanisms are being adopted, rather than attempting to produce totally new malware. All types of malware such as worms, rootkits viruses, script viruses, trojans, macro viruses, backdoors, spyware, key loggers, etc. are being recycled to produce new variants of old malware. Symantec Internet Security threat published report in 2011 (Symantec Enterprise Security 2011b) and 2010 (Symantec Enterprise Security 2011a) announced that the malicious code activity continues to grow at a record pace, and there are over 2.8 million new malicious code signatures, mostly developed in 2009. Other sources show the escalating infection rates with almost 120 million servers identified to be infected in the first quarter of 2010, with 64% of which were attacked by unknown malicious code (Komisarczuk 2010). Recently, McAfee Labs identified almost 60,000 new pieces of malware per day and showing the sophistication in malware that makes their detection very difficult (Alperovitch 2011).

In 2004 Marx revealed that the AV engines need an average of ten hours to respond with a publicly available update (Marx 2004). The spread of malware instances can be extremely fast, with some infections requiring only a few seconds (Staniford *et al.* 2004). In December 1999, experiments conducted by the San Diego Supercomputer Center, where an operating system was installed and connected to the internet with no security updates, the computer was attacked just within eight hours of installation, this contrast with an attack every 39 seconds (General Information Security Statistics 2004).

After 21 days of installation, the system had experienced 20 different attacks and within about 40 days the computer had been compromised. In another case, when PSINet Europe purposely built an unprotected server and connected it to the Internet in 2003, their results were astonishing: in the first 24 hours the server was maliciously attacked 467 times and a total of 626 malicious attacks were recorded over the three week period (Eisner 2003; James 2007). More recently, these figures have grown exponentially (Alperovitch 2011; Maria 2011; Yao & Liu 2011).

In summary, new malware with variants or obfuscations from existing known malware are generated rapidly and are used to attack systems that are vulnerable to inflict as many systems as possible before AV companies are able to take any countermeasure. Hence, it is a very important requirement of a robust malware detection technique to handle obfuscating transformations. Sections 2.3 to 2.6 provide a literature survey of various commonly adopted detection techniques.

2.3 Conventional Malware Detection

Malware detectors are used to scan a computer system to identify malware, with the main purpose of preventing it from adversely affecting the system. In order to protect computers AV engine must perform three main tasks: Scanning, Detection, Removal (Bakshi *et al.* 2010). A Malware detector (D) is defined as a function whose domain and range are the set of executable program (p), and its duty to determine executable program is malicious or benign D: P → malicious, benign. Modern and traditional AV engines scan the programs (p) in a system for a byte sequence or malware signature (s) that matches with the stored database engine. If a signature is found in the program (p), it

will be identified as a malware, otherwise it is declared as benign, and this is represented in the equation 2.1.

$$D(P) = \begin{cases} MALware \ \ if \ s \in p \\ Benign \ \ otherwise \end{cases} \tag{2.1}$$

The malware signature is a byte sequence (a number derived from a string of text) that uniquely identifies a specific malware. An example malware signature for VirusWin32.Bolzano malware is shown in Figure 2.1 (a) Virus.Win32.Bolzano's source code (b) Virus.Win32.Bolzano's signature. Typically, a malware detector uses the malware signature to identify the malware like a fingerprint. Most countermeasures such as anti-malware engines are supplied with a database containing information of existing malware in order to identify maliciousness by looking for code signatures or byte sequences while scanning the system (Marx 2004; Townsend 2010). A malware detector scans the system in various locations for characteristic byte sequences or signatures that match with the one in the database and declares existence of malware, and subsequently blocks its access to the system. The process is called signature-based detection and most traditional AV engines use this method (Chouchane & Lakhotia 2006). It is a very efficient and effective method to detect known malware. The major drawback of this method is the inability to detect new or unknown malicious code and zero day attacks (Sung *et al.* 2004; Xu *et al.* 2004). Therefore, updating the detection engine or AV software daily with latest malware signatures is essential so as to protect the computer system against all known malware. The more malware signatures are fed into the AV engine, the more effective it is in detecting latest known malware. Since hidden and obfuscated malware apply sophisticated evasion techniques, signature-based AV engines

fail to detect them. Symantec software has announced in its websitethat LiveUpdate is the most trusted way of updating virus definitions, but not for unknown malware (Symantec Enterprise Security 2011a, 2011b).

The new threat for computers is that the malware writers can change the byte sequence of the malcode without affecting the objective of the code, by using obfuscation techniques such as packing, polymorphic transformations and metamorphic obfuscations, instead of creating an entirely new malware (Tang *et al.* 2010). Signature based AV scanners will not be able to detect these new malware due to the non-existence of their fingerprints in the signature database. Hence, there is a need to capture the behaviour of malware based on anomalies or behavioural patterns exhibited by such hidden malware, which is the main focus of this research work (Chandola *et al.* 2009).

Due to the growing size of the malware signature database, many AV software developers and researchers working on malware detection have suggested major changes in the scanning algorithm such as, i) using an expert system based on Artificial Immune System (AIS) method (Dasgupta 1997), ii) using I/O Request Package (IRP) sequences (Zhang *et al.* 2010), iii) using protocol-level malware scanner (Shetty 2004) to scan data that is being transferred or downloaded to a computer system at the protocol level, iv) using disk processor to monitor disk requests to identify malicious programs based on characteristic properties of the disk requests (Paul 2008), and v) using scanning of multiple parts of a file to perform diffraction analysis for detecting polymorphic malware similar to X-RAY scanning (Perriot & Ferrie 2004).

Virus.Win32.Bolzano's source code	
B8 F2070000	MOV EAX,7F2
03C5	ADD EAX,EBP
E8 0D000000	CALL 0041158D
B8 07080000	MOV EAX,807
03C5	ADD EAX,EBP
E8 01000000	CALL 0041158D
C3	RETN

(a) Win32.Bolzano's source code

Virus.Win32.Bolzano's signature
B8F2 0700 0003 C5E8 0D00 0000
B807 0800 0003 C5E8 0100 0000 C3

Figure 2.1 (a) Win32.Bolzano's source code (b) Win32.Bolzano's signature

(b) Win32.Bolzano's signature

Although signature based detection is one of the well-established methods by AV engines, there is a need to resort to the above mentioned behaviour based methods as signature based detection suffers from the following drawbacks:

- ***High false positive***: This occurs by identifying benign files as malware. In May, 2007, Symantec updated their malware signatures and crippled thousands of Chinese PCs by mistakenly identifying two core Windows.dll files as Trojan horse that Symantec dubbed "Backdoor.Haxdoor" (Keizer 2007).

- ***High false negative***: from failing to detect unknown or new malware (Paul 2008).

Even though the quality of malware detectors used in popular AV software and anti-forensic methods is improving in their techniques, from virus signature-based detection towards heuristic-based detection, the malware cyber criminals are one step ahead (Ghosh & Turrini 2010; Venkatraman 2010).

2.4 Modern Detection

Countermeasures such as detection engines are responsible for detecting malicious code and classify the detected code based on the effects of that code to fall under categories such as viruses, worms, Trojans, spywares, adware, etc. Different detection engines such as the traditional signature based detection and heuristic based detection have evolved (Daoud *et al.* 2008). Malicious authors, in order to avoid signature based detection approaches, adopt a number of stealth techniques. Due to the inability of traditional signature based detection approach to detect obfuscated malicious code, the focus of research has shifted to improve heuristic based detection (TreadwellZhou & Zhou 2009).

2.4.1 Heuristics Based Detection

Heuristic methods can be static or dynamic, with recent malware detectors using signature based detection along with heuristics to detect variants of existing this heuristics approach is dependent on the behaviour of the malware (Symantec Enterprise Security 1997). Usually, malware authors design their malicious code to achieve a set of malicious functions; therefore, each piece of malware is unique. Heuristics approaches use certain base rules that determine the proper functioning of the system, its stability and data integrity.

Heuristic detection is successful in detecting new malware that are generated by applying toolkits on the known viruses, worms or Trojans to generate thousands of new malware from the same malware, each one distinct but belonging to the same family. Hence, heuristics based detection is good to identify threats belonging to the same family and useful for detecting macro viruses (Konstantinou & Wolthusen 2008). For instance, when 'NewVirus' comes out, the definition created to detect it will also successfully identify 'NewVius.b', 'NewVirus.c', 'NewVirus.d', and so on. For example, the Vundo Trojan that is designed to drop Adware onto a compromised system, has several family members. Symantec categorises the Trojan into two distinct families, in 2007 Trojan.Vundo (Symantec Enterprise Security 2007), and in 2009 they were realized to be from the same family and the new malware was termed Trojan.Vundo.B. (Symantec Enterprise Security 2009b). The only issue of this detector is the false positive rate is high, where it could identify a non-malicious file as a malware.

Researchers have proposed a heuristic detection approach that targets obfuscated windows binary files being loaded into memory researchers (TreadwellZhou & Zhou 2009). They look for anomalies, such as Original Entry Point (OEP) with Entry Point code that starts with a JMP or CALL, or suspicious imports from KERNEL32, or multiple PE Headers, or Patched Import table that has incorrect calculated SizeOfCodem, SizeOfData and SizeOfImage in the header. All of these methods could be used by malware authors and heuristics based detection can detect them within the existing malware families.

Due to the high false alarm rate, heuristics based detection is believed to work well when combined with another detection technique such as signature based detection

(Yang *et al.* 2010). Results of other detection techniques used to support heuristics based detection (Konstantinou & Wolthusen 2008). However, a malware author could write malicious code that does not fire the rules to cause damage.

2.4.2 Behavioral Based Detection

Behavioural detection is the recent trend for anti-virus softwares and does not rely on signatures to detect malware. This detection is a dynamic analysis technique that observes the behaviour of a program by executing it in a sand-box environment in order to analyze the file during runtime. Whenever the behaviour of the malware seems "suspicious", it is flagged as malware and action will be taken. Behaviour based detection requires a templates or profile of suspicious behaviour (Govindaraju 2010a, 2010b). In other words, the templates or profile of the malware becomes its signature. Thus behaviour based detection technique is a kind of signature based detector except that the signature here is the functionality of the malware. This detection has two detection schemas (Jacob *et al.* 2008). First is Passive detection, which scans computer files to see if there are any deviations from the normal profile. Second is Active detection, which uses a sandbox to monitor the behaviour of a program.

It is important to mention here that the recent malware is using code obfuscation methods that combine static evasion along with dynamic evasion techniques. However, the issue of the behaviour based detection approach is the high false positives, where a non-malicious file identified as malware is also not desired. The false positives are due to the fact that it is usually difficult to define malicious behaviours in an accurate way (Fukushima *et al.* 2010).

2.4.3 Semantic Based Detection

A semantic analysis technique is used when the malcode is responsible to determine the malicious nature of the code. A signature is created based on the semantic property of the code (Christodorescu *et al.* 2005; Preda *et al.* 2008). The idea of this detection method is used for creating signature based on program functionality, and not based on the byte sequence of the program.

In 2005 Christodorescu et al. state that the fundamental deficiency in the pattern-matching approach to malware detection is that it is purely syntactic and ignores the semantics of instructions (Christodorescu *et al.* 2005). Therefore, they proposed semantics-aware malware and put forward a malware detector that is able to handle some of the obfuscations commonly used by hackers. Experimental evaluation has shown that semantic detection can detect variants of malware with a relatively low run-time overhead.

Since code obfuscation techniques, as shown in Section 2.6, change the malware signature but not its behaviour, which has to be preserved, formal methods for program analysis, such as semantics-based static analysis and model checking, could be used in designing more sophisticated malware detection algorithms (Kong *et al.* 2010). Semantic based methods are able to deal with obfuscated versions of the same malware, such as identifying similarities through their execution traces (Preda *et al.* 2008). However, malware developed using code transposition and instruction substitution techniques can still evade semantic based detection methods.

2.4.4 Hidden Markov Model Based Detection

Hidden Markov models (HMMs) have been extensively used in biological sequence analysis as it is well suited for statistical pattern analysis. Since 1970, it is used to analyze and understand a Markov process and provide a result based on a series of observations related to the process (Krogh 1998). HMM is a state machine where the transitions between states have fixed probabilities, and it relies on the current situation and does not consider any past situation.

In recent years, it has been shown that the detection of metamorphic malware is very effective using Markov models applied to malware detection (Wing Wong 2006). In addition, it is also been used to determine malware family (Camastra *et al.* 2011).

Hidden Markov models provide a means to describe sequence variations statistically. On the other hand, Profile Hidden Markov Models (PHMM) is known for their success in biological sequence analysis. It has been found that PHMM can effectively detect metamorphic malware as well (Attaluri, S. & McGhee 2009; Govindaraju 2010a, 2010b).

In 2007 and 2009, Attaluri showed that PHMM can be successfully used to detecting metamorphic malware, they still need to use machine learning concept of having good and better accuracy (Attaluri, Srilatha 2007; Attaluri, S. & McGhee 2009).

2.4.5 Similarity Analysis

Similarity analysis is a detection method based on the analysis of similarities of distance measures. Distance measures play an important role in a vector model, and similarity

analysis can be performed by using three commonly used distance measures, namely Euclidean, Manhattan and cosine similarity measure for nearest neighbor (NN) that is primarily used in text mining. In short, the maliciousness of a code is estimated (Shabtai *et al.* 2009). For instance, malware such as Win32.Evol (Orr 2006) has a multiple variant of the same sample malware because of the obfuscation methods. Similarity based detection approach can be used between the variants to check whether the variant is the child of the sample under inspection (Alazab 2011). Understanding the relationship used among the distance measures can help us to choose a proper distance measure for malware detection.

Usually similarity detection is conducted by performing static analysis, where the executable program is first disassembled using reverse engineering tools. Each disassembled executable (**P**) and the variant disassembled executable (**P'**) represent a vector of functions x, y, each function is represented as an array of vector of functions. The similarity between the functions of a program P and P' is computed. The value is then compared with the threshold value to determine if the executable is malicious or not.

2.5 Malware Analysis

Detection techniques can be in static, dynamic or hybrid forms. Static analysis uses the syntax and structural properties of the file. Dynamic analysis is also called Process Under Inspection (PUI) method, which means the analysis of the files during its running time. Hybrid analysis combines static with dynamic analysis. Theoretically, a static analysis is very effective on the information captured from structural properties, like sequence of bytes "signatures" and anomalies in file content. Since dynamic analysis is only effective

with runtime information, such as running process of the PUI, and these are usually evaded by hidden malware, in this research the focus is on static anomaly based detection with the prime objective of identifying hidden malware (Egele *et al.* 2012).

Even though live analysis techniques could help in capturing evidence during forensic investigations to a certain extent, they are far from infallible and lead to false negatives of hidden malware. In live analysis, malware such as rootkits can hide and change itself without being seen. Moreover the attackers can target hidden area on the system structure to hide the malware.

Static analysis techniques analyze the code of a program without executing it. To perform static analysis on binary code, the binary code is required to be disassembled first, in other words converted into corresponding assembler instructions. Next, conclusions about the program behaviour can be derived by applying various control flow and data flow analysis techniques. The advantage is static analysis can exhaustively analyse the complete program code by examining all possible paths of execution. It is usually faster than the dynamic analysis as behaviour analysis is quite time consuming in dynamic analysis, where the system state keeps changing. However, the main drawback of static analysis is that attackers can deliberately craft malware that are hard to analyze statically. In particular, they can make use of various code obfuscation techniques to confuse the disassembly and code analysis. This problem is being addressed in this research work.

In (Islam *et al.* 2010) authors used pattern recognition algorithms and statistical methods, theirs framework combines the static features of function length and printable

string information extracted from malware samples into a single test of 1400 unpacked malware and 151 clean files, they achieved an overall classification accuracy of over 98%. Same authors in theirs study (Tian *et al.* 2010) to distinguish malicious files from benign files on a dataset of 1368 malware and 456 benign files by investigating the behavioural features using logs of various API calls, using runtime features of malware files, the experimental results provided an accuracy of over 97%.

2.6 Code Obfuscation

Code obfuscation is used to transform the program code in such a way to make it difficult to read and understand. Detecting malware is a game of obfuscation and de-obfuscation that malware writers and AV vendors play against each other. The malware writers use obfuscators to evade detection, while, AV vendors try to deobfuscate code and improve their detection techniques. Recently, this game has become more advanced and highly complicated for AV vendors, preventing them from de-obfuscating the code easily. One such successful approach being adopted by malware writers is the evasion technique such as polymorphism (Szor 2005; Townsend 2010), metamorphism (Wing Wong 2006) and packing (Sun *et al.* 2010), where the malware is able to morph code such that detection techniques fail. Packers, metamorphic and polymorphic malware use command sequences that can be altered into a program without changing the main behaviour to evade the scanning process of the signature detection of malware. Example os this type of malware include MetaPHOR, Win32/Simile (MetaPHOR 2010), Lexotan32 (Orr 2007), W32.Evol (Orr 2006), RPME and Mistfall / Zmist (Ferrie & Szor 2001; Szor 2005). These are advanced obfuscated malware that demonstrate a set of packer, polymorphic

and metamorphic code writing skills, which include entry-point obscuring, randomly using an additional polymorph decryptor, code permutation and code integration.

The term 'obfuscation' is defined here to mean the modifying of program code in a way that keeps it functionally identical with the aim to reduce vulnerability to any kind of static analysis, to deter reverse engineering and make it difficult to understand, and less readable. Obfuscation techniques such as packing, polymorphism and metamorphism are used by malware authors as well as legitimate software developers. They both use code obfuscation techniques for different reasons. Using code obfuscation is very effective to malware author to evade the antivirus scanner since it modifies the program code to produce offspring copies which have the same functionality but with different byte sequence to make sure the new code is not recognized by antivirus scanner. By looking at real world threats such as Win32/Parite, Win32/Rimecud, Win32/Alcan, Win32/Rbot, Win32/CeeInject, Win32/Nachi Win32/Bagle, and many others, it has been found that the malware writers are recycling existing malware with different signatures by using obfuscation techniques such as packing, polymorphic transformations and metamorphic obfuscations, instead of creating an entirely new malware (Paul 2008). Because of such threats, sophisticated malware detection techniques are required, especially those that can capture the behaviour of malware based on observed anomalies.

Obfuscation methods could transform malcode into a new code without affecting the original functionality or purpose so that the AV engine’s scanning process skips the detection of the signature. The VX Heavens website provides access to thousands of thousands of malware variants in a variety of different categories (VX Heavens 2011).

For each malware variant, a signature must be identified, packaged, and downloaded to the signature database of users expecting protection from the new attack.

Malware authors are continually developing new techniques for creating and applying obfuscation techniques T(p) on a malware program (p) to produce an obfuscated program (p') as shown in the equation 2.2

$$T(P) = P' \tag{2.2}$$

Thereby making it very difficult to reserve engineer and decipher the signature successfully, even though the two programs p and p' have the same functionality and exhibit the same affect. On the other hand, since p and p' have different byte sequence, AV engines and reverse engineers are applying de-obfuscation techniques D(p') on the obfuscated program (p') in order to analyse the malware and to detect the malware, as shown in Figure 2.2. In summary, malware authors use obfuscation techniques to defeat the signature based detection by changing the malware signature. These techniques are described next.

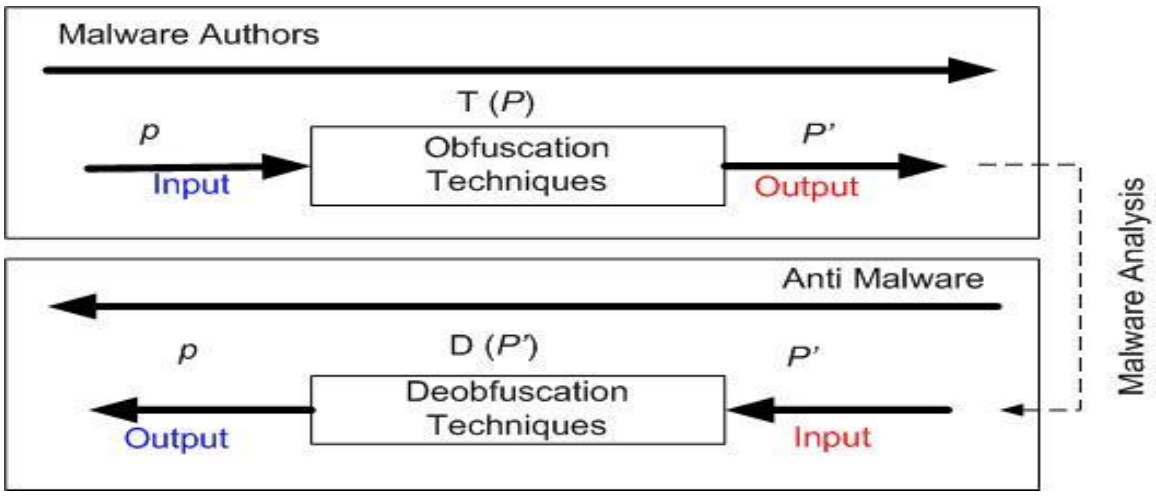


Figure 2.2 Obfuscation Transformation

2.6.1 Packing

Recently, malware authors have used packers to avoid detection and to run malware faster. This results in mainly changing any byte sequence in the PE into a different byte sequence in the newly produced packed PE. Packing the malware makes the obfuscation method difficult to understand (Sun *et al.* 2010) and the malware authors only need to change a small number of lines of code in order to change the malware signature.

Packers are commonly used today for code obfuscation or compression. Packers are software programs that could be used to compress and encrypt the PE in secondary memory and to restore the original executable image when loaded into main memory (RAM). Cyber criminals do not need to change several lines of code to change the malware signature mainly because, changing any byte sequence in the PE results in a new different byte sequence in the newly produced packed PE and starts from its original entry point (OEP) so that the challenge here to find the OEP. For instance, Themida[i], Obsidium[ii], ASPack / ASProtect[iii], PECompact[iv], and Armadillo[v] are all commonly used packers and malicious code authors are using such packers to produced new codes. Figure 2.3 explain three different packer protections (a) PECompact (b) Themida (c) ASPack. Packers have the essential features of reducing the size of malware, making malware easier to transform, and thereby producing malware more resistant to static analysis. Hence, packers being able to bypass detection engines have become the most favorite toolkits.

[i] www.oreans.com
[ii] www.obsidium.de
[iii] http://www.aspack.com
[iv] http://www.bitsum.com/pecompact.php
[v] www.siliconrealms.com

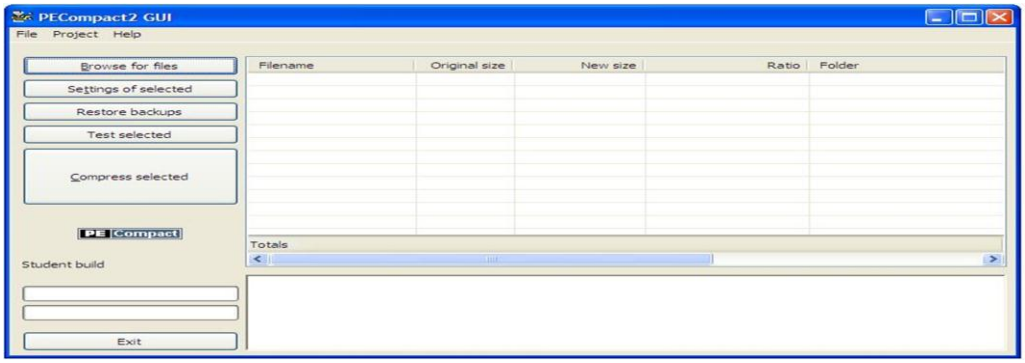
2.6.2 Polymorphic Malware

Polymorphic malware uses encryption to change the body of the malware, and changes decryption routines in each infection to avoid static byte sequence and as long as the encryption keys change, the malware becomes difficult to detect by anti-malware programs (Tang *et al.* 2010; Xu *et al.* 2004). This has led AV programs to use different scanning techniques, from simple byte sequence matching to more complex techniques such as X-RAYING scanning (Perriot & Ferrie 2004), which scan multiple parts of a file and performs different analysis to detect polymorphic malware. Polymorphic malware is hard to detect through signature based detection since it could change the byte sequence on its own. Such examples are s W32.Fujacks, W32.Vundo, P2P-Worm.Win32.Polip, Virus.DOS.Chameleon, and w32.Detnat (Gu *et al.* 2007; Li *et al.* 2011). Figure 2.4 shows a polymorphic code example of P2P-Worm.Win32.Polip.

Polymorphic technique enables a malicious program to mutate at byte level when the program creates a copy of itself, where every new copy of the malware is encrypted with a unique key, which contains a unique byte sequence. Anti-malware vendors are confronting a serious problem of defeating the complexity of malware. Polymorphic malware uses encryption and data appending/ data pre-pending in order to change the body of the malware, and further, it changes decryption routines from infection to infection as long as the encryption keys change, making it very difficult to create antivirus signatures to block infections. Crime-ware tool kits such as CRUM Cryptor Polymorphic, PoisonIvy Polymorphic Online Builder and Mariposa, use polymorphic code and obfuscation techniques to avoid detection, and are available in black-market

with updates for between $50- $10000 depending on the features included (Alazab, Watters, *et al.* 2011). As a result, this will lead to anti-malware experts developing different scanning techniques from simple byte sequence matching to a combination of difficult antivirus engines to block its numerous propagation techniques. In early 2011, Symantec Internet Security Threat Report (Symantec Enterprise Security 2011a, 2011b) stated that detecting polymorphic malware such as w32.Polip and w32.Detnat is much more difficult and complex than any other type of Malware. The uses of simple scanners have made this type of obfuscation prolific and pose to continue the threat.

Countermeasures such as AV engines defeat the polymorphic methods by waiting for the malware to de-obfuscate itself. Malware detection system can run a malware in a sandbox or emulators (Balakrishnan & Schulze 2005) in order not to cause any damages or affect the system. As the malware executes, it must decrypt itself to attempt to infect a file and therefore the emulator can scan for the malware signature when the malware decides to execute. However, this technique can still thwart the detection engine and the encryption algorithms have become sophisticated and complex that it becomes very hard to detect such malware in real time. These types of new obfuscation mechanisms such as Web-attack toolkits continue to drive up the number of malware variants in common circulation. In 2010, Symantec encountered more than 286 million unique variants of malware (Symantec Enterprise Security 2011a).



(a) Packer protection from PECompact

(b) Packer protection from Themida

(c) Packer protection from ASPack

Figure 2.3 Different Packer Protections (a) PECompact (b) Themida (c) ASPack

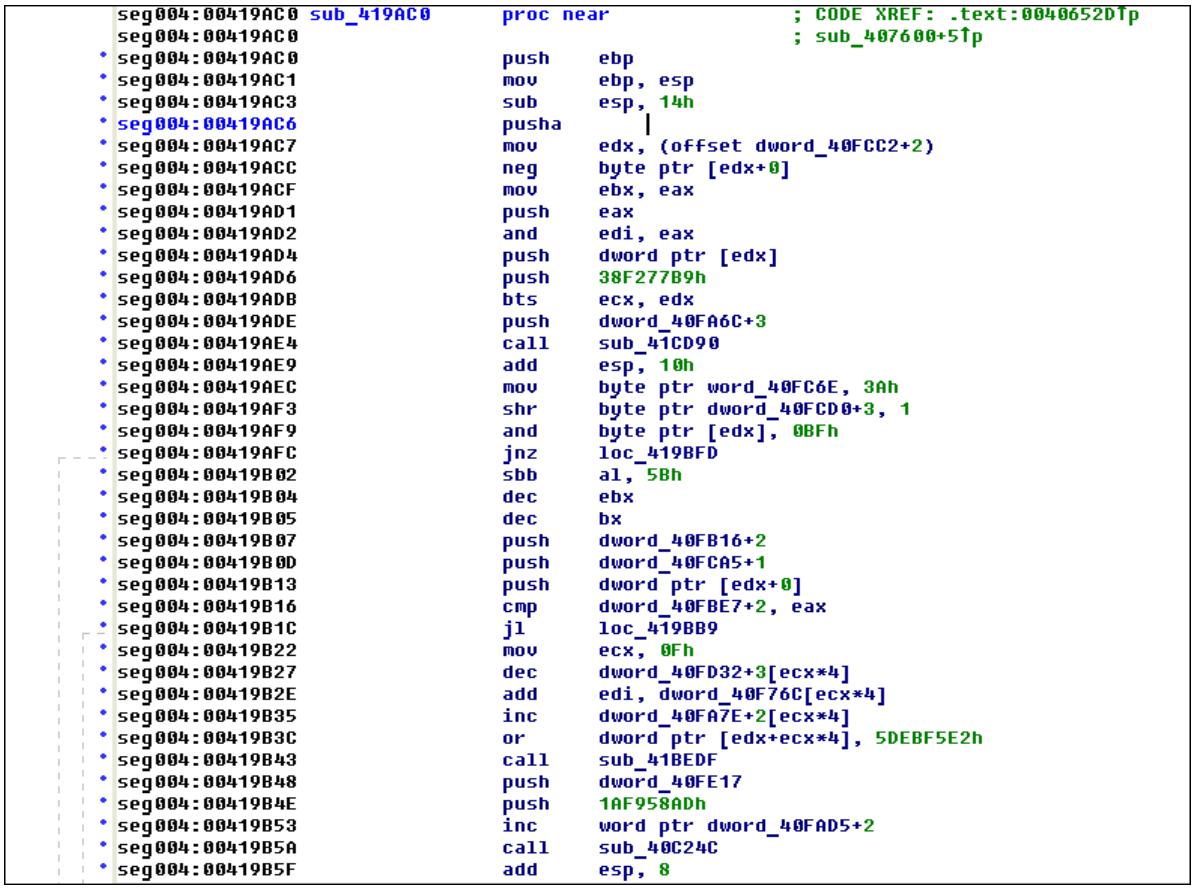


Figure 2.4 The Polymorphic Code Example of P2P-Worm.Win32.Polip

The creations of polymorphic malware toolkits such as Mutation Engine, Dark Angel's Multiple Encryptor, NuKE Encryption Device, and TridenT Polymorphic Engine, have created big challenges for security researchers and AV engines (Li *et al.* 2011). Generally the polymorphic malware toolkits engine, a small object file linked with the malware that would make a new polymorphic virus. The code entitles the users to provide their own random number generator or can use the default one. Also, these toolkits provide a documentation of the engine which describes how it could be used and it also includes a demonstrator malware using the engine.

2.6.3 Metamorphic Malware

Malware authors resort to sophisticated hiding techniques. Metamorphic engine uses code obfuscation techniques to produce morphed copies of an original program (Desai 2010; You & Yim 2010). It changes the code itself without the need of using encryption. In general, there are four techniques commonly used for metamorphic obfuscation. These are,

i) *Dead-code Insertion* which inserts operation that do nothing, such as a sequence of **NOP**s (No Operation Performed),

ii) *Code Transposition* which changes the instruction sequence, such as using **JMP**s instructions so that the order of instructions is different from the original one,

iii) *Register Reassignment* such as replacing [PUSH ebx] with [PUSH eax] to exchange register names, and

iv) *Instruction Substitution* which replaces the instructions with different instructions so as to have the same result - some authors uses a database dictionary of equivalent instruction sequences to make it easier and faster. Subsections explain each one in details.

The main difference between polymorphic and metamorphic is in the malware body. In polymorphic, the malware is encrypted by different encryption key during each infection and the malware body will decrypt by the same code across multiple infections. Metamorphic could use many techniques to transforming the code as shown in the

subsections below. Win95.Regswap was one of the early metamorphic viruses to make an impact via register usage exchange (Aycock 2006; Bakshi *et al.* 2010; Desai 2010; You & Yim 2010). Figure 2.5 shows the original code of Virus.Win95.Regswap. An example of metamorphic techniques used on the original virus code is described in the subsection that follows.

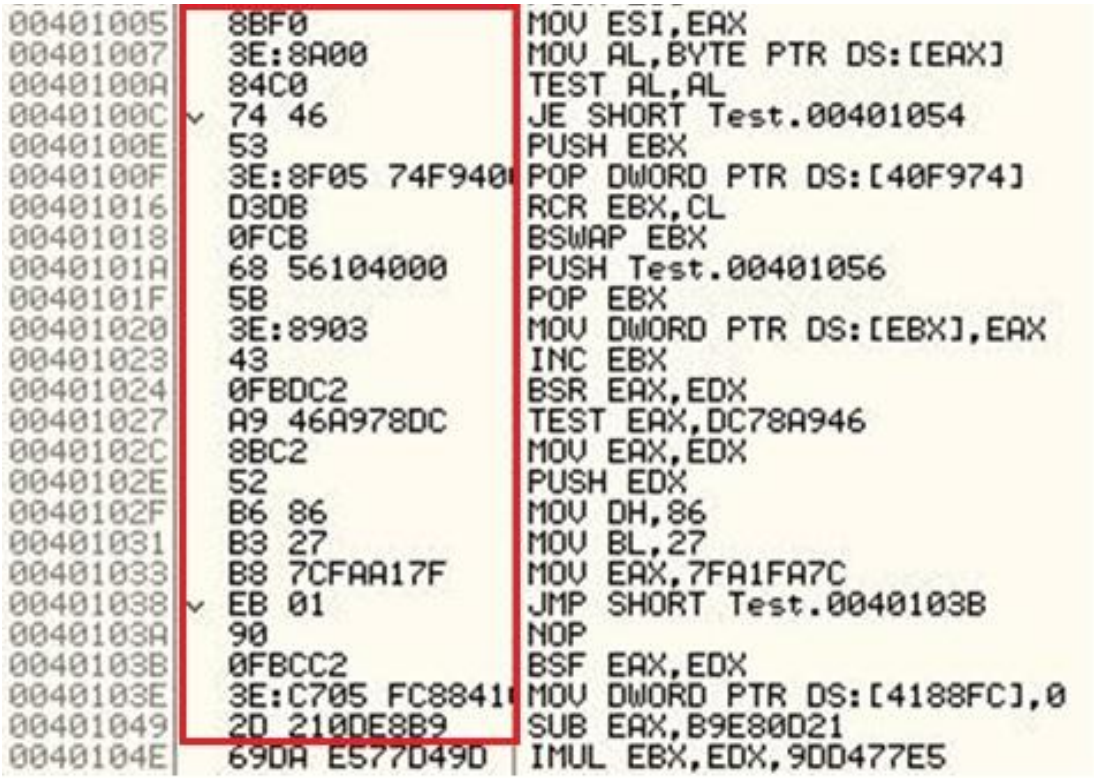


Figure 2.5 Original code of Virus.Win95.Regswap

2.6.3.1 Dead Code Insertion

Dead-code insertion also known as trash insertion, involves insertion of code that does not change the malware behaviour such as a sequence of NOPs (No Operation Performed). Dead codes are designed to evade detection and fool antivirus software that

use basic signature-based detection matching. This is illustrated in Figure 2.6 (You & Yim 2010).

Dead-code Insertion, which does nothing to the code logic but change the byte string of the code can be difficult to detect such as changes in the code using complicated code of sequence of operations if the function or code has no effect, used to change the virus signature to some extent. Examples; [NOPs], [MOV eax, eax], [SHL eax, 0], [ADD eax, 0] and [INC eax] followed by [DEC eax], not only, also passing values through memory rather than registers. Using such tricks of dead code insertion can make the analysis time consuming and sometimes fail the detection engine to detect such threat(Christodorescu & Jha 2004).

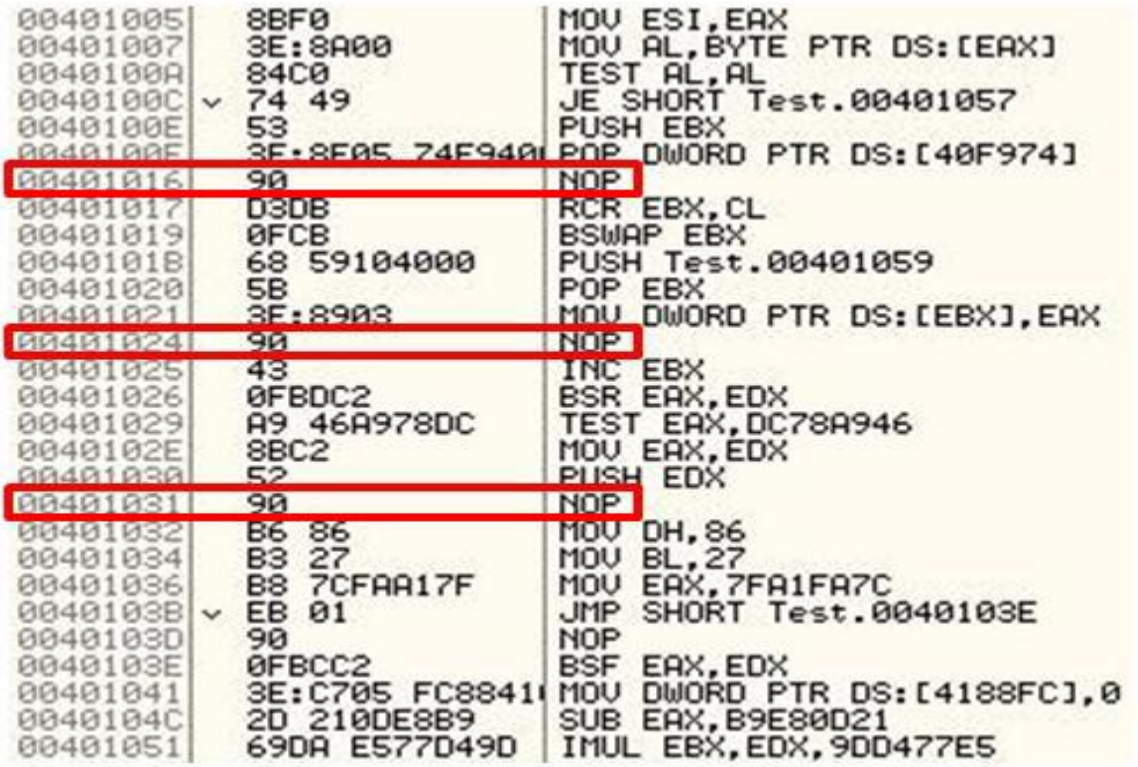


Figure 2.6 Dead–Code Insertion

2.6.3.2 Code Transposition

Code transposition shuffles the instructions so that the order in the binary content is completely different resulting in a new signature than the one used by the antivirus software and this could evade detection. Transposition use *Jump* instructions to shuffle the binary content. Figure 3.3.2 shows an example of code transposition. There are some scenarios that show how to detect this type of obfuscated such as, reorder the instructions and insert unconditional branches, or *jump* instructions to restore the original control. This is then followed by swap instructions if they are not interdependent. Most analysis techniques use an intermediate representation, such as the Control Flow Graph (CFG) (Bruschi *et al.* 2006) or the Program Dependence Graph (PDG) (Ferrante *et al.* 1987), that is robust against superfluous changes in control flow.

Code Transposition such as Subroutine Permutation, Subroutine Inlining and Subroutine Outlining, results in changing of the instructions such that the order of instructions is different than the parent code, with the use of instructions such as using JMP, CALL instructions as shown in Figure 2.7 and Figure 2.8 (You & Yim 2010).

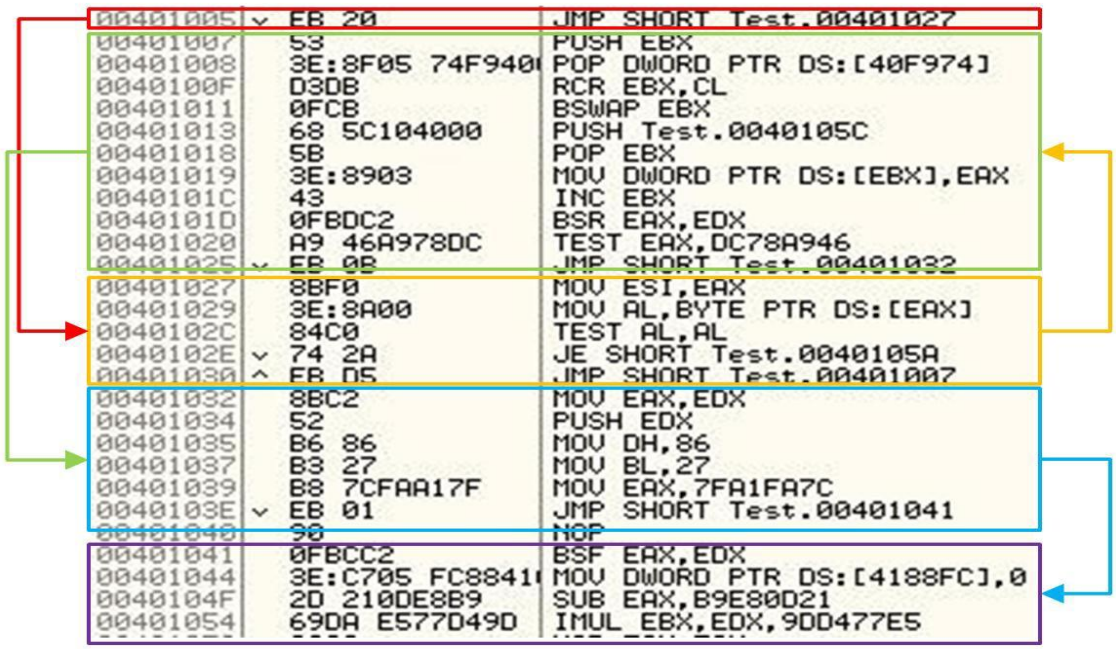


Figure 2.7 Code Transposition based on Unconditional Branches

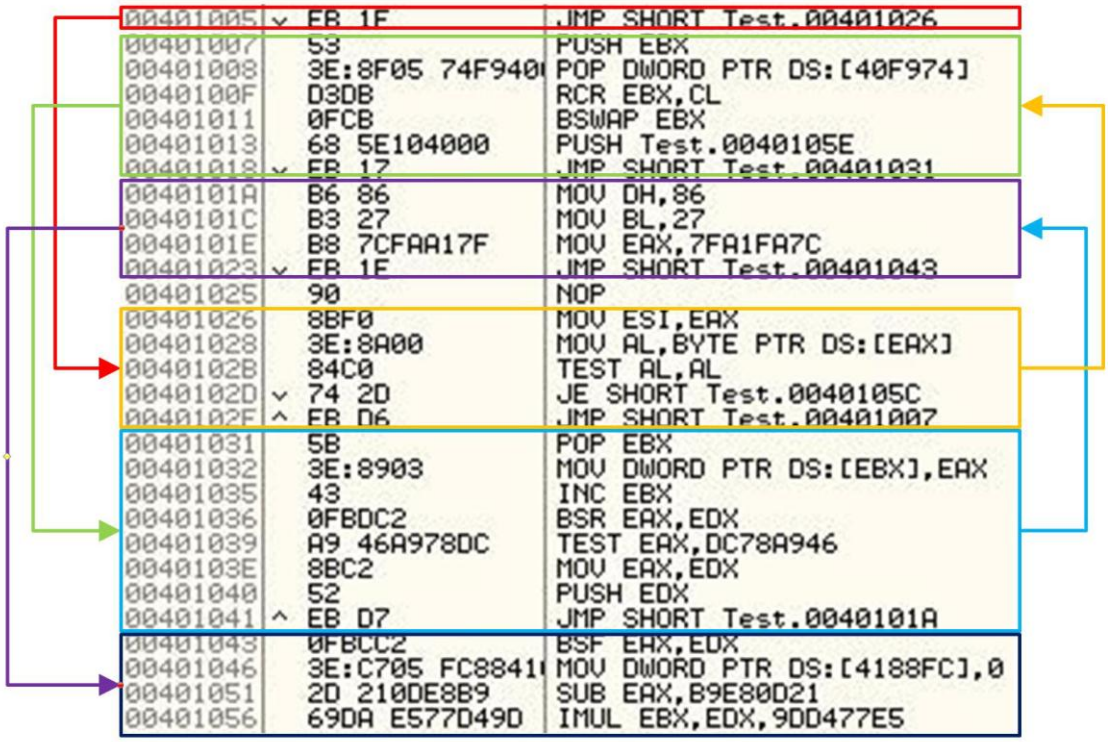


Figure 2.8 Code Transposition based on Independent Instructions

2.6.3.3 Register Reassignment

The register reassignment transformation replaces code between registers by exchanging register names with no other effect on program behaviour. For example, if register ebx is dead throughout a given live range of the register eax, it can replace eax in that live range. The signature that encodes [PUSH ebx] is not the same as the one that encodes [PUSH eax] and hence becomes obfuscated. Register Reassignment such as replacing [PUSH ebx] with [PUSH eax] to exchange register names is shown in Figure 2.9.

Figure 2.9 Register Reassignment

2.6.3.4 Instruction Substitution

This obfuscation technique uses a dictionary of equivalent instruction sequences to replace one instruction sequence with another. Figure 2.10 shows an example of

instruction substitution (You & Yim 2010). This kind of obfuscation relies upon human knowledge of equivalent instructions, making it a challenge for automatic detection of malicious code. For example, the IA-32 instruction set is rich and contains many different ways to perform the same operation. The IA-32 assembly language provides ample opportunity for instruction substitution. Instruction substitution uses an equivalent code substitution to replace the instructions by different instructions with the same result. Some authors use a database dictionary of equivalent instruction sequences to make it easier and faster, as shown in Table 2.1.

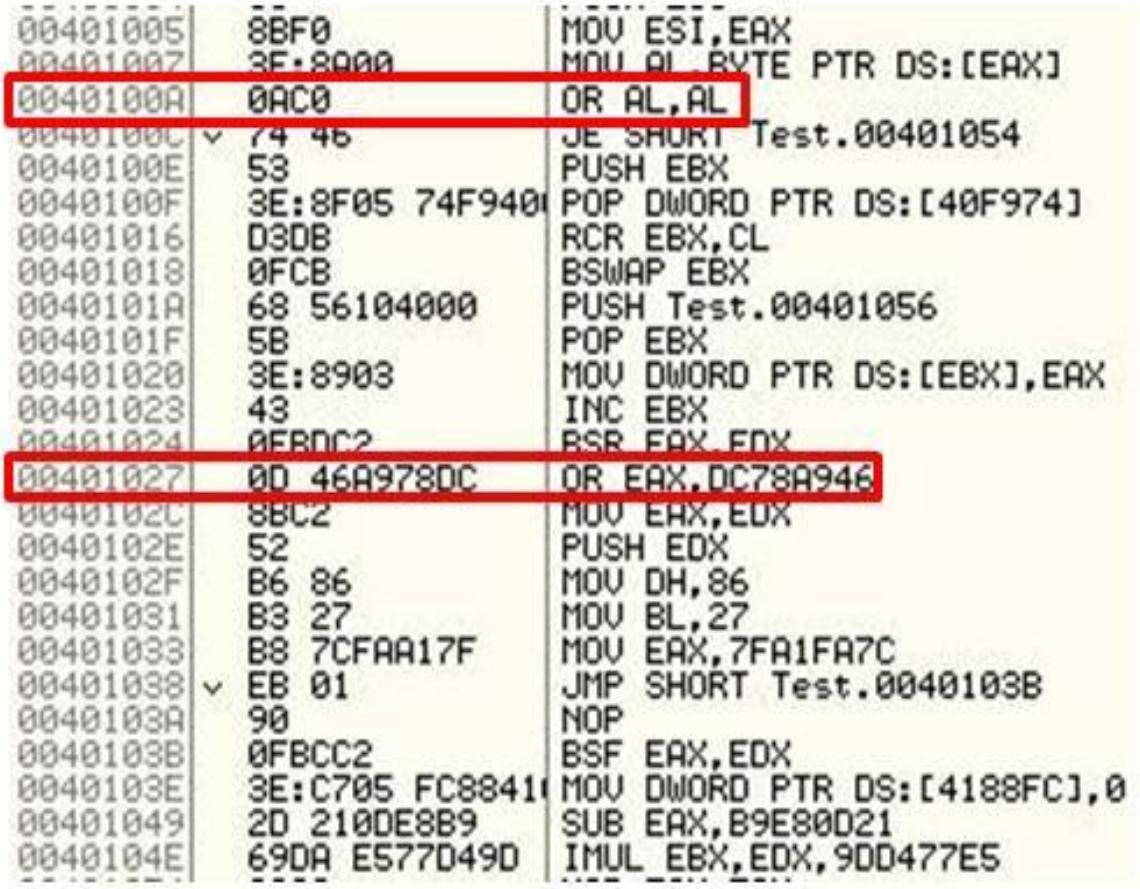


Figure 2.10 Instruction substitution.

Table 2.1 Instruction substitution and an equivalent code substitution

Instructions	Equivalent
MOV EAX,ECX	XOR EBX,EAX
MOV EBX,EAX	XOR EAX,EBX
MOV ECX,EBX	XOR EBX,EAX
XOR EAX, EAX	MOV EAX, 0
MOV EAX, IMM	PUSH IMM POP EAX

OP REG1,REG2	MOV MEM, REG1
	OP MEM, REG2
	MOV REG, MEM

2.7 Summary of Literature

In summary, signature based detection is disadvantageous as it cannot be used to detect novel attacks and the repository of known signatures has to be continually augmented to include newer signatures. Evidence has shown that new attacks are frequently generated through modifications of known attacks. The present malware detection system usually rely on existing malware signatures with limited heuristics and are unable to detect those malware that can hide itself during the scanning process in online systems. In this dissertation, will be focus on develop a robust digital forensic process for NTFS file system, and then to design and apply innovative techniques for fulfilling the main objective of detecting hidden malware, also propose effective digital forensic techniques that could be used to analyse and acquire evidences of hidden malware in NTFS disk images is very important.

Literature studies on malware detection have shown that there is no single technique that could detect all types of malware and countermeasures cannot detect

unknown malware or unknown signatures which are uniquely identify a specific malware (Christodorescu & Jha 2004; Skoudis & Zeltser 2003). Therefore, signature based approaches fail to detect unknown malware. On the other hand, anomaly-based detection uses the knowledge of normal behaviour patterns to decide the maliciousness of a program code. It has the key advantage and ability to detect zero day attacks. However, it is very difficult to accurately specify the system or program's behaviour and thus these approaches usually are resulting in more false positives (Kolter & Maloof 2006; Symantec Enterprise Security 2011b).

As shown in the advanced malware, the malware obfuscation technologies have become sophisticated and complex. Clearly, such a tendency is expected to be retained based on the growth of the hardware and software technologies. Also, they will be revised to be suited for the popular infrastructures such as web and smartphone.

The purpose of the research is a positive step towards overcoming the digital forensic problems identified in the above sections. The possible approaches and methods that would be adopted in this research are detailed in Chapter 3, Chapter 4, Chapter 5 and Chapter 6. In a nutshell, these methods are categorised under 2 parts, i) Digital Data Investigation of Slack Space (Physical) Chapter 3 and ii) Obfuscated Malware Detection (Logical) Chapter 4, Chapter 5 and Chapter 6. The first part involved investigation of the hidden malware in the slack space for identifying known and unknown malware as malware attackers take advantage of the NTFS weaknesses and the inability of existing AV to check the slack space. In the second part, a statistical n-gram analysis of binary content that is based on operation codes (op-codes) and API system call sequence feature together with the data mining learning techniques such as support vector machine (SVM)

algorithms were used to classify whether the binary content is benign or malicious These proposed methods are quite novel and would help in identifying hidden malware with a focus on an improved anomaly-based detection.

Chapter 3 : Forensic Analysis of the NTFS File System

SHERLOCK HOLMES: *"The world is full of obvious things which nobody by any chance ever observes".*

—Sir Arthur Conan Doyle, *"The Hound of the Baskervilles,"*

The Strand Magazine (1902)

"Not everything that is undocumented is automatically useful…

Some operating system internals are just internals in theirs

Strict scene, that is, implementation details."

— Sven Schrieber

3.1 NTFS File System

Forensic analysis of the Windows NT File System (NTFS) could provide useful information leading towards malware detection and presentation of digital evidence for the court of law. Since NTFS records every event of the system, forensic tools are required to process an enormous amount of information related to the user / kernel environment, buffer overflows, trace conditions, network stack and many more. This has led to forensic tools that are imperfect and though they are commercially available, they are not comprehensive and effective (Richard & Roussev 2006). Many existing techniques have failed to identify malicious code in hidden data of the NTFS disk image (Vassil 2009). This chapter discusses the analysis technique explored to successfully

detect maliciousness in hidden data, by investigating the NTFS boot sector. The chapter also reports the experimental studies conducted with some of the existing popular forensics tools and their limitations that have been identified. Further, through the proposed three-stage forensic analysis process, the chapter shows how the experimental investigation has attempted to unearth the vulnerabilities of NTFS disk image and the weaknesses of the current forensic techniques.

3.2 NTFS Investigation Goal

Digital investigation is a process to answer questions about the compromised computers (Carrier 2005). This chapter focuses on static analysis since it can capture all the hidden information that cannot be modified during the analysis process, unlike live analysis techniques which could result in having falsified data. An image copy of the NTFS hard disk would capture hidden data and hence this study entails developing efficient techniques to analyse this data in a confined lab environment for the identification of hidden malware. This research work forms an important contribution in digital forensics, which is the science of identifying, extracting, analysing and presenting the digital evidence that has been stored in the digital electronic storage devices to be used in a court of law (Baryamureeba & Tushabe 2006; Kruse & Heiser 2001; Reed & Angel 2007). It takes an initial step towards addressing the open problem of identifying unseen or new malware that could evade detection in the form of hidden or obfuscated malicious code in the physical file system.

Static analysis of NTFS file system which is the standard and most commonly used file system could provide useful information for digital forensics. This chapter

discusses the analysis technique explored to detect data hidden based on the internal structure of the NTFS file system in the boot sector. Further, it attempts to unearth the vulnerabilities within a NTFS file system disk image and highlights the weaknesses of the current forensic techniques.

The main goal of this chapter is to propose a methodology to effectively detect hidden malware stored in the physical NTFS file system. Since NTFS file system is predominantly used in most computer systems, and malware attackers take advantage of weaknesses in NTFS to hide malware, this chapter focuses on hidden malware forensic. With this in view, in this chapter the research aims i) to explore the NTFS disk structure and its vulnerabilities, ii) to investigate weaknesses of existing commonly used digital forensic techniques such as signature-based, heuristic-based and anomaly-based, and iii) to propose and evaluate improved methods in static analysis of NTFS for identifying hidden malware by investigating the physical disk image.

A further goal of this chapter is to effectively detect hidden malware that could be implanted on a computer system in the slack space (physical) on hard disk drives. Hence, this chapter provides the background and methods adopted in investigating the physical slack space and in identifying the hidden malware. To facilitate the research process, a good understanding of the NTFS file system is essential, and this is briefly described next.

3.3 Windows Architecture

Operating systems play a key role in identifying illegal activities. As files and their structure are determined by the operating system, understanding the relationship between the files and the files system of the operating system and their communication mechanism is really important for digital forensics.

3.3.1 NTFS File System Architecture

Today, NTFS file system is the basis of predominant operating systems in use, such as Windows 2000, Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista, and Windows 7 and supported by Linux based distributions. Due to the widespread use of the NTFS files system, attackers try to target NTFS, as this could result in affecting more computer users. Another compelling reason for witnessing a strong relationship between computer crime and the NTFS file system is the sparse studies in literature that unearth the vulnerabilities of NTFS and lack of standardization in digital forensics procedures and techniques (Palmer 2001; Rogers & Seigfried 2004).

The key feature to note in NTFS disk structure is that the Master File Table (MFT) is the core of NTFS file system since it contains details of every file and folder on the volume and allocates two sectors for every MFT entry. The MFT entry within the MFT contains attributes that can have any format and any size. Further, as shown in Figure 3.1 every attribute contains an entry header which is allocated in the first 42 bytes of a file record, and it contains an attribute header and attributes content. The attribute header is used to identify the size, name and the flag value. If the size is less than 700

bytes (known as a resident attribute), the attribute content will reside in the MFT followed by the attribute header, otherwise it will store the attribute content in an external cluster called cluster run known as a non-resident attribute. This is because; the MFT entry is 1KB in size and hence cannot fit anything that occupies more than 700 bytes. In addition, since the Windows operating system does not zero the slack space, it becomes a vehicle to hide data, especially in $Boot file.

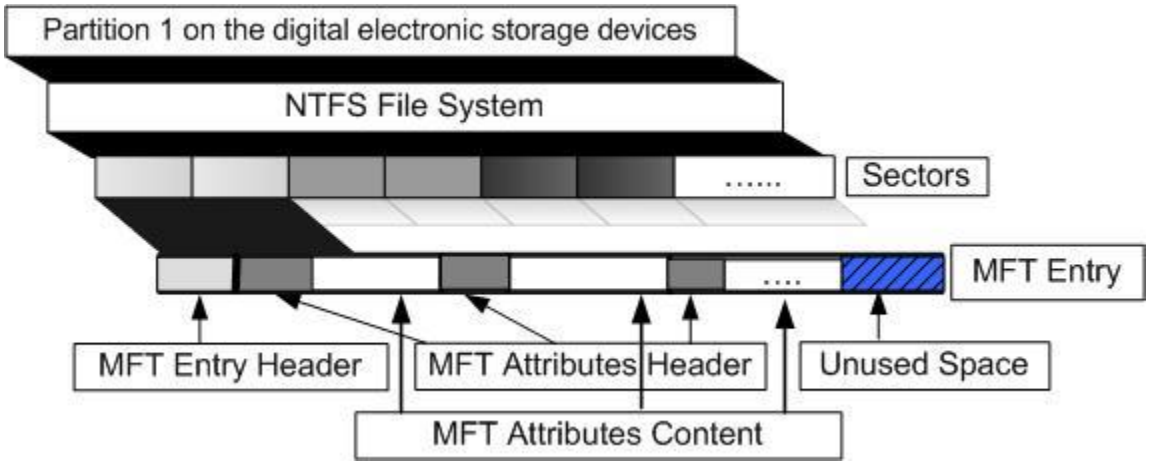


Figure 3.1 MFT Layout Structure

In this chapter a preliminary investigation conducted in these aspects revealed the following problems that form the primary motivation of this research work. In an operating system, the file system is responsible the organization method of data on a hard disk volume. It comprises of two parts: the collection of files that store related data, and a directory structure that organizes and provides information about all the files in the system. The two most popular file systems used by windows users of today are FAT32 (File Allocation Table) and NTFS file system. Later this chapter will discuss the NTFS

file system more in details and how it can allow malicious code to be hidden away from the radar of any commercially available antivirus tool.

3.3.2 Portable Executable Architecture

The Win32 Portable Executable File Format (PE) introduced by Microsoft is the standard executable format for all versions of the operating systems on all supported processors. Therefore, the approach explored in this research would be tested directly on the Portable Executable (PE) format files. Most programs in windows are constructed by accessing the Windows application programming interface (API) through functions available in dynamic Link Library (DLL) on the system. Microsoft provides a great number of DLLs, and each DLL can be used by more than one program at the same time.

For the obfuscated malware detection system, this research focuses on extracting feature from API sequence and how to automate the API calls' extraction process. Therefore, it is important to understand the Portable Executable (PE) structure as the data structures adopted on disk are the same as the data structures used in the memory. For example, when the function LoadLibrary is called to load the executable into memory, a data structure such as the IMAGE_NT_HEADERS is identical on disk and in the memory. However, the Windows loader looks at the PE file and decides what portions of the file to be mapped in. This mapping is consistent in that higher offsets in the file correspond to higher memory addresses when mapped into memory. Since a PE file comprises of various sections and headers, free PE tools (PE Explorer) has been used to view, analyze and reverse engineer Windows PE files such as EXE, DLL, OCX, ActiveX

Controls, Screensavers, SYS, DPL, Control Panel Applets, MSSTYLES, BPL, and executable files that run on MS Windows platform.

Table 3.1 provides a good understanding of the PE structure that includes DOS headers and PE headers. The PE header starts with the signature bits “PE” and has some file properties, such as timestamp, machine type and the number of sections. Section Table has code sections (.text), and data sections (.data). The .text section is the default section for code and the .data section stores writable global variables. It also contains the file’s Original Entry Point (OEP). OEP refers to the execution entry point of a PE file, where the file execution begins. Finally, the .rdata section contains read-only data.

Traditional detection engines search the binary files for stored signatures to identify known malware. However, the malware can easily fool the detection engine by applying obfuscation technique such as Packer (section 2.6.1) to obfuscate the malware internal code and data structures from being detected by security software.

Table 3.1 Portable Execution Structure

Portable execution Structure
Section Table
DOS Header
COFF File Header
Optional Header
Standard fields
NT additional fields
Optional Header Data Directories
Export Table
Import Table
Resource Table
Exception Table
Certificate Table
Base Relocation Table
Debug
Architecture
Global Ptr
TLS Table
Load Config Table
Bound Import
Import Address Table (IAT)
Delay Import Descriptor
COM+ Runtime Header
Reserved
Section Table
.text
.rdata
.data
.idata
.rsrc

Metafile has been defined and its importance explained: Metafile refers to the file structure of the NTFS file system. It is also used to define files, system driver management volume, buffer file system changes, assign a drive letter to each partition, manage free space allocation, and store security and disk space usage information. The metafiles are treated specially by Windows and are difficult to directly view them. Metafiles in the NTFS disk root directory start with "$" character, and it is hard to get information about them by standard means. By looking at $MFT file size, it is possible to find out useful information such as, time spent by the operating system in cataloguing the entire disk. Table 3.2 provides the metadata files used by the operating system and their functions are described.

3.4 Digital Crime Investigation Analysis

According to Michael Andrew (Andrew 2007), the general components in overall digital forensic process are; acquisition, preservation, and analysis. On similar notes, Brian Carrier (Carrie 2003; Carrier 2005) state that there is a three-process approach for digital investigation, which consists of system preservation, evidence searching, and reconstruction processes. Towards achieving the goal to detect hidden data based on the internal structure of the NTFS file system in the boot sector, this chapters describes the three stages used to perform digital forensic analysis in a comprehensive manner. The three stages as explain in Figure 3.2 are; Stage 1: Hard disk data acquisition, Stage 2: Evidence searching, and Stage 3: Analysis of NTFS file system.

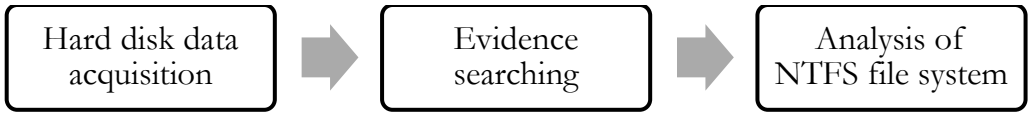


Figure 3.2 The Three Stages of a Digital Crime Investigation

'Hard disk data acquisition' is the process of preserving the state of the digital crime scene and the purpose of this stage is to maintain the original evidence without being overwritten during the investigation work. Such a process ensures that evidence is not lost in the investigation process and also to cater to the need for this stage to undergo any future analysis that may be required. The 'Evidence searching' stage mainly involves searching for data that support the hypotheses about the incident, and details about how to perform searching inside the files and file system of the NTFS in a secure way. In the Analysis of NTFS file system stage, existing evidence collected, the physical place in the hard disk, unreached place such as slack space or between the partition, etc. are analysed. Therefore, it is very important to understand Medium Data and the volume in hard drives, and these are described in Sections (3.4.1) and (3.4.2).

This chapter attempts to identify and fill the gap found in literature and practice by studying the techniques used in the analysis of the NTFS disk image. Hence, the main objectives are i) to explore the NTFS disk image structure and its vulnerabilities, ii) to investigate different commonly used digital forensic techniques such as signatures, data hiding, timestamp, etc. and their weaknesses, and iii) finally to suggest improvements in static analysis of NTFS disk image.

Table 3.2 NTFS Metadata Files Information

Metadata Name	Metadata	#	Description
Master File Table	$MFT	0	Itself MFT
Master File Table Mirror	$MFTmirr	1	Copy of the first 16 MFT records placed in the middle of the disk or the end of the partition.
Log File	$LogFile	2	Transaction logging file for the volume.
Volume Decription Table	$Volume	3	Contains information about the volume label (partitions), file system version, time, etc.
Attributate Definition Table	$AttrDef	4	List of standard files attributes on the volume
Root Directory	$.	5	Root directory
Cluster Allocation Bitmap	$Bitmap	6	Volume free space bitmap
Volume Boot Code	$Boot	7	Boot sector (bootable partition)
Quota Table	$Quota	9	Containing information if disk quota are being used on the volume. (only for NTFS)
Upper Case Table	$Upcase	10	Table containing information for coverating file names to the Unicode 16 bit file naming system for international.

3.4.1 Medium Data Analysis

Some computers use big-endian ordering and others use little-endian ordering of digital data to organize multiple bytes (sets of 8 bits, 0 or 1) and the most common technique used is ASCII or Unicode to encode the characters. A sector represents the basic unit of data storage on a hard disk with each sector being able to store 512 bytes of data (4096 bits). All types of Microsoft OS are designed to read and write blocks of data called clusters and these clusters are made up of an even number of sectors that are fixed blocks of data, e.g. 1024 bytes (2 sectors), 4096 bytes (8 sectors), etc. The number of sectors needed for a cluster is dependent upon the type of storage device, the operating system involved and the size of the logical storage device.

The data collected for analysis from a computer system connected to a network could be classified as volatile and nonvolatile data. Volatile data is information that might be lost if the system is turned off, such as the current network connection, running services open TCP or UDP ports, internal routing table, users logged on, open files, running processes, scheduled jobs, cached NetBIOS name table, and others. To analyse volatile data, also called Live Analysis, the system needs to be running during the investigation. On the other hand, nonvolatile data is information that is not lost if the system turn off such as register data, file system time and date stamps, system version and patch level. In a Linux based system, information such as file system MD5 checksum values, user accounts, IIS logs, stored files could be investigated to collect evidence of malware. Such analysis of nonvolatile data is called Static Analysis.

Figure 3.3 shows the different analysis areas on a computer system, and physical storage media analysis forms the base and lowest level analysis of devices such as hard disks, memory chips, CD-ROMs, Flash memory, and others. Since all of the data is stored on medium devices, and that malware exploits the weakness of the disk structure to store itself without being recognized by the AV engines, the necessity emerges to start analysis from the physical storage media. Therefore, analysis of the physical storage media could provide useful information leading towards malware detection and presentation of digital evidence for the court of law.

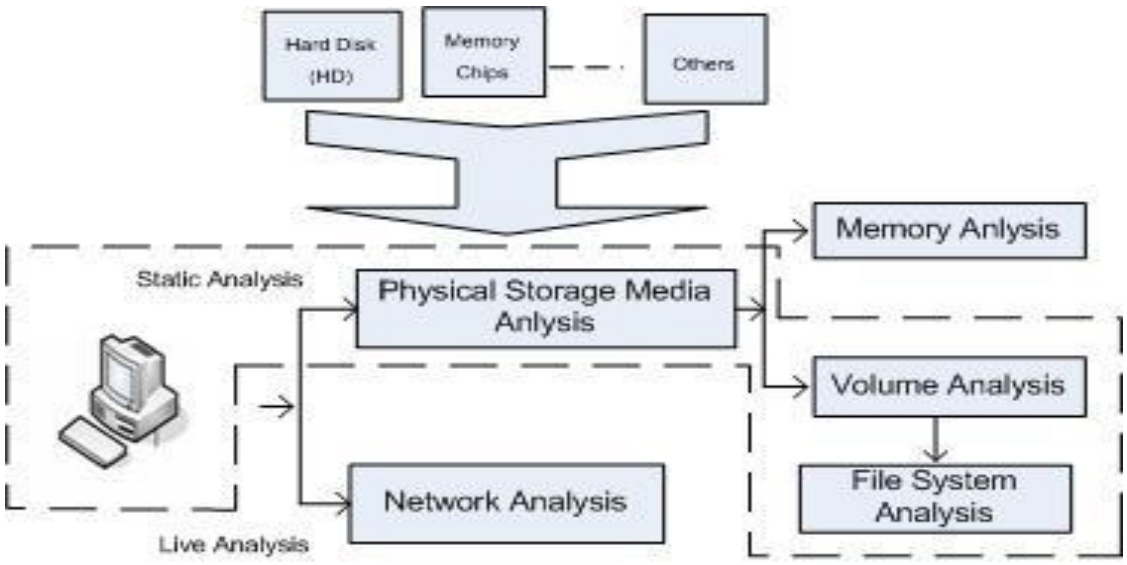


Figure 3.3 Analysis Areas

3.4.2 Volume Analysis

A volume is a collection of addressable sectors that the users can use for data storage (read and write) and is responsible for the creation of partitions that could be defined as a set of consecutive sectors. Figure 3.4 provides an example of HDD partition and volume

of a typical NTFS file system. Volume Analysis involves looking at the data structure with partitioning and assembling data in digital storage media devices that are stored when installing Microsoft Windows operating system. Examples of such information are, the primary or logical partitions on the hard disk, and in the windows system, each partition has one or more tables to describe the starting and the ending sector (the length) and the type of partition. Even though these concepts are found to be similar in other operating systems such as the Linux operating systems, not all of them use volume in the same way as Microsoft Windows does.

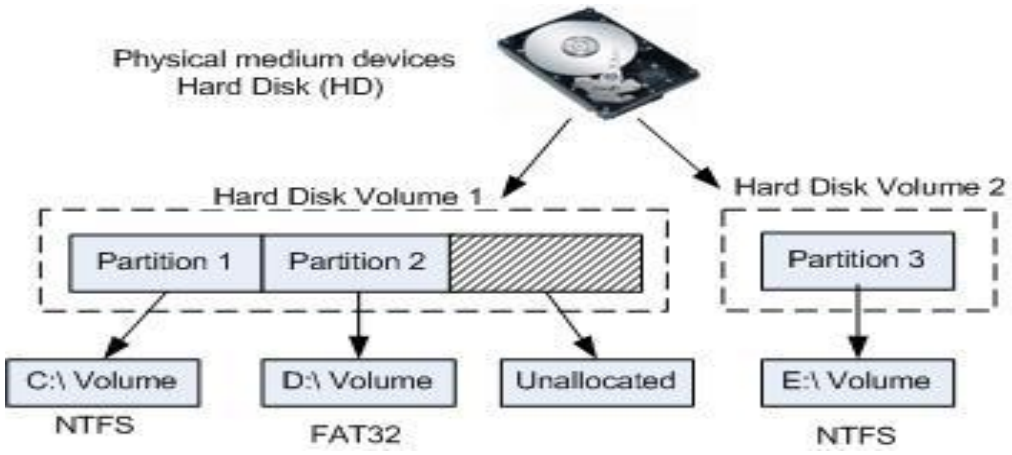


Figure 3.4 Hard Disk Drive Volume and Partition.

Partitioning is dependent on the OS and not the type of interface on the hard disk. Windows OS uses the same partition system, whether the disk uses an AT Attachment interface (ATA/IDE) or Small Computer System Interface (SCSI). This research project is not focusing on the hard disks and types but their basic structures are explored to see how the malware can hide itself as hidden data. Many existing techniques have failed to identify malicious code in hidden data of the disk structure (Naiqi *et al.* 2008). Therefore it is very important to analyse the volume system and the first task is to investigate the

data structure that describes the volumes. When analysing a volume system it is important to check the consistency of the partitions by checking the start and end sectors.

Disassembly is the process of recovering a symbolic representation of a program from its binary representation. Disassembling binaries is a difficult task for two primary reasons: variable-length instructions and fundamentally indistinguishable data embedded inside code regions. There are two standard disassembly techniques, the linear sweep method and the recursive traversal method. Various techniques have been proposed to improve the disassembly coverage and accuracy. Cifuentes et al. in (Cifuentes & Fraboulet 1997; Cifuentes & Gough 1995) used speculative disassembly techniques to improve disassembly coverage. Their approach made certain assumptions on the properties of the machine and the conventions of the programming language or the operating system. Program obfuscation has been used to protect software security. For example, it has been used to protect software content from malicious reverse engineering (Sun *et al.* 2010; You & Yim 2010). Linn and Debray (Linn & Debray 2003) proposed two techniques to foil disassemblers by tailoring the static analysis process against a particular tool. One is through inserting unreachable junk code into the program and the other is through using a branching function in place of regular call instructions. Kruegel et al. (Kruegel *et al.* 2004) proposed static binary analysis techniques to improve the disassembly success rate for obfuscated binaries. These static disassembly methods cannot correctly handle code which contain self-modifying and overlapping instructions or in which other static-resilient techniques are used in (Zhang, 2009). Additionally, to apply these various techniques to disassemble code buried in network traffic, the starting

point of the code should be found first. Our choice reverse engineering tool IDA pro uses the linear sweep method.

3.5 Problem Background

With the recent trend in malware to operate as hidden malicious code that evade detection by anti-virus tools and other live analysis techniques, this research firstly attempts to identify hidden malware that adopt Physical such as “Slack Space” implantation techniques through exploiting the vulnerabilities of the system .

3.5.1 Vulnerabilities of NTFS Disk Structure

Cybercriminals make use of file system vulnerabilities in order to infect more computers and guarantee effectiveness of evading security countermeasures. For instance, keeping the last modified date of an infected file unchanged to make it seem like it was uninfected was one of the first early techniques cyber criminals had adopted to thwart detection. The cybercrimes are based on exploiting the vulnerability in applications and operating systems. This type of matured obfuscation technique has resulted in hidden class of threats that many would not have been exposed to previously.

Cyber criminals target a vulnerable area on the system structure to hide the malware. Since NTFS is predominantly used in most computer systems, and malware cyber criminals take advantage of NTFS weaknesses to hide malware, more computers get infected without being detected by commercial detection engines (Naiqi *et al.* 2008). They are capitalizing on the vulnerabilities of NTFS to hide the malware from AV engines and further exploit the weaknesses of the present digital forensic techniques from

being detected. A preliminary study conducted on the hidden data of the $Boot file (Alazab, Venkataraman, *et al.* 2009; Alazab, Venkatraman, *et al.* 2009; Huebner *et al.* 2006). It takes an enormous amount of time to analyse the data derived with such tools and most of the existing tools are complex and not easy to use. Moreover, not all computer infections are detected by forensic tools, especially intrusions that are in the form of hidden data in the $Boot file go unchecked. Hence, the conclusion drawn here is that the existing forensic tools are not comprehensive and effective in identifying the recent computer threats. This confirms what is reported in literature as well (Ahmad 2002; Palmer 2001; Purcell & Lang 2008; Reith *et al.* 2002).

As shown in Section 3.3.1 the architecture of the NTFS file system have weaknesses which led to attackers using different techniques such as disguising file names, hiding attributes and deleting files to intrude the system.

NTFS, Windows NT's native file system, is designed to be more robust and secure than other Microsoft file systems. The key feature to note in NTFS disk structure is that the Master File Table (MFT) contains details of every file and folder on the volume and allocates two sectors for every MFT entry. Since the Windows operating system does not zero the slack space, cybercriminals make use of MFT to hide malicious code without raising any suspicion. Such limitations in NTFS have led to cyber criminals using different techniques such as disguising file names, hiding attributes and using deleted files to intrude the system.

3.5.2 Insufficiency of Malware Detection Tools

Current live malware detection tools such as anti-virus software are able to identify known malware. Once new malware is released, the AV engines will reactively update their signatures to combat the new malware. However, recent methods adopted by computer intruders, attackers and malware are to target hidden and deleted data so that they could evade from virus scanners. As a result, some malware adopt circumvention techniques such as polymorphic, metamorphic obfuscations, etc. so that they cannot be detected through current live analysis techniques.

Countermeasures such as detection engines are failing to detect malware and are identifying benign file as malware resulting in high false alarm rate and false positives. In 2010 David Stang tested 41 updated scanners on 54,016 malware files, finding less than 1 among the files capable of being detected by only half of the scanners used and on an average the scanner detected only 62% with a maximum of about 80% (Stang 2010). However, sophistication in malware through code obfuscation has created another challenge for digital forensic examiners, namely the detection rate of new and unknown malware is very low, (Passerini *et al.* 2009; Symantec Enterprise Security 1997), and identifying benign code as malicious (false alarm rate) is quite high (Stang 2010). This is due to the fact that existing techniques and methods do not perform sufficient statistical analyses to determine if the anomaly was 'actually' malicious. Though recent studies that use statistical analysis of file binary content including statistical n-gram modeling techniques (Shafiq *et al.* 2008; Stolfo *et al.* 2005, 2007) concentrate on identifying malcode in document files, the statistical modeling of hidden malcode that predominantly

use Windows API calling sequence that reflects the behaviour of a particular piece of code to evade detection is yet to be explored.

3.5.3 Weaknesses in Digital Forensic Analysis Tools

There is no standard methodology and approach for conducting digital forensics investigations (Palmer 2001), in spite of the fact that digital crimes are on the rise and less than 2% of the reported cases result in conviction (Baryamureeba & Tushabe 2006). The literature surveys conducted in the last few years on digital forensics were more focused on the technical aspects without any consideration for a generalized model. While some road maps indicate steps to collect image copy of physical devices, the main challenge is that "analytical procedures and protocols are not standardized nor do practitioners and researchers use standard terminology" (Palmer 2001). This has resulted in a variety of forensic analysis tools that provide different ways to search for the digital evidence of malware and most of these tools adopt different techniques for different kind of information.

Since NTFS file system stores all events that take place on a computer system, huge amount of data analysis is required while scanning the entire NTFS disk image for forensic purposes. From a preliminary study of this research work conducted on the hidden data of the $Boot file, it is observed that a variety of tools and utilities have to be adopted along with manual inspections to identify unseen malware.

3.6 Proposed Forensic Analysis Process

Since the physical medium is the fingerprint of both legal and illegal activities, it is very important to analyse the physical medium for malware deposits. This research adopts a procedure of data acquisition from NTFS images as explaining in Section 3.8.1. There are currently no consistent or standardized procedure for accomplishing digital forensics and there are many models and frameworks suggested by many organisations, agencies and researchers such as: Farmer and Venema (Farmer & Venema 2005), Kruse and Heiser (Kruse & Heiser 2001), III and Roussev (Richard & Roussev 2006) , The Enhanced Integrated Digital Investigation Process (EIDIP) model by Baryamureeba and Tushabe (Baryamureeba & Tushabe 2006) and the enhanced version of the Integrated Digital Investigation Process Model (IDIP) proposed by Brian Carrier and Eugene Spafford (Carrier & Spafford 2003), the Abstract Digital Forensics Model (Reith *et al.* 2002),Forensic Process Model by the National Institute of Justice (NIJ), the Scientific Working Group on Digital Evidence (SWGDE 2000).

In this section, the forensic analysis process proposed and adopted to achieve the above mentioned objectives of this research work are described. An empirical study conducted using selected digital forensic tools that are predominantly used in practice is explained. Several factors such as effectiveness, uniqueness and robustness in analysing NTFS disk image have been considered in selecting the tools / utilities required for this empirical study. Each utility does some specific functionality, a collection of such tools have been adopted to perform a comprehensive set of functionalities. The forensic

utilities / tools used to conduct the experimental investigation in this research work are listed below:

i. Disk imaging utilities such as DD (Garner 1970) or DCFLDD (Harbour 2006) for obtaining sector-by-sector mirror image of the disk;

ii. Evidence collection using utilities such as Hexedit (Phillips 2010), Frhed 1.4.0 (Kibria 2009) and Strings V2.41 (Russinovich 2009) to introspect the binary code of the NTFS disk image;

iii. NTFS disk analysis using software tools such as The Sleuth KIT (TSK) v3.01 (Carrier 2011), Autopsy (Carrier 2010) and NTFSINFO v1.0 (Russinovich 2006) to explore and extract intruded data as well as hidden data for performing forensic analysis.

In the first stage of forensic analysis, the DCFLDD developed by Nicholas Harbour (Harbour 2006) and DD utility from George Garner (Garner 1970) was used to acquire the NTFS disk image from the digital electronic storage device. This utility was selected for investigation since it provides simple and flexible acquisition tools. The main advantage of using these tools is that one could extract the data in or between partitions to a separate file for more analysis. In addition, this utility provides built-in MD5 hashing features. Some of its salient features allow the analyst to calculate, save, and verify the MD5 hash values. In digital forensic analysis, using a hashing technique is important to ensure data integrity and to identify which values of data have been maliciously changed as well as to explore known data objects.

Stage 1: Hard disk data acquisition,

Stage 2: Evidence searching and

Stage 3: Analysis of NTFS files system.

3.6.1 Stage 1 - Hard Disk Data Acquisition

In the first stage of forensic analysis, the DCFLDD developed by Nicholas Harbour (Harbour 2006) and DD utility from George Garner (Garner 1970) was used to acquire the NTFS disk image from the digital electronic storage device. This utility was selected for investigation since it provides simple and flexible acquisition tools. The main advantage of using these tools is that one could extract the data in or between partitions to a separate file for more analysis. In addition, this utility provides built-in MD5 hashing features. Some of its salient features allow the analyst to calculate, save, and verify the MD5 hash values. In digital forensic analysis, using a hashing technique is important to ensure data integrity and to identify which values of data have been maliciously changed as well as to explore known data objects.

3.6.2 Stage 2 - Evidence Searching

The next stage involved searching for evidences with respect to system tampering. An evidence of intrusion could be gained by looking for some known signatures, timestamps as well as even searching for hidden data. In this stage, the Strings command by Mark Russinovich (Russinovich 2009), Frhed hexeditor tool by Rihan Kibria (Kibria 2009) and WinHex hexeditor tool by X-Ways Software Technology AG (X-Ways Software Technology AG) we used to detect a keyword or phrase from the disk image.

3.6.3 Stage 3 - Analysis of NTFS File System

In the final stage of the experimental study, the data obtained from the NTFS disk image were analysed, that contributed towards meaningful conclusions of the forensic investigation. A collection of tools were adopted such as The Sleuth Kit (TSK) (Carrier 2011), Autopsy Forensic by Brian Carrier (Carrier 2010) and NTFSINFO from Microsoft Sysinternals by Mark Russinovich (Russinovich 2006) to perform different aspects of the NTFS file system analysis.

3.7 Forensic Investigation Steps

Many aspects must be taken into consideration when conducting a computer forensic investigation (Venkatraman 2011). There are different approaches adopted by an investigator while examining a crime scene. From the literature, five steps are commonly adopted, such as, Policy and procedure development, Evidence assessment, Evidence acquisition, Evidence examination, and Documenting and reporting. In the proposed approach for the digital forensic investigation described in this chapter, the following nine steps as shown in Figure 3.5are suggested and were successfully adopted in the experimental study:

Step 1: Policy and Procedure Development – In this step, suitable tools that are needed in the digital scene are determined as part of administrative considerations. All aspects of policy and procedure development are considered to determine the mission statement, skills and knowledge, funding, personal requirement, evidence handling and support from management.

Step 2: Hard Disk Acquisition – This step involves forensic duplication that could be achieved by obtaining NTFS image of the original disk using DD tool command. This step is for obtaining sector-by-sector mirror image of the disk and the output of the image file is created as Image.dd.

Step 3: Check the Data Integrity – This step ensures the integrity of data acquired through reporting of a hash function. The MD5 tool is used to guarantee the integrity of the original media and the resulting image file.

Step 4: Extract MFT in the Boot Sector – In this step, the MFT is extracted from the boot sector. The MFT is analyzed using WinHex hexeditor tool, and NTFSINO is used to check the number of sectors allocated to the NTFS file system.

Step 5: Extract $Boot file and Backup boot sector – In this step, the $Boot file is extracted to investigate hidden data. The hidden data is analyzed in the $Boot metadata file system using WinHex, TSK and Autopsy tools.

Step 6: Compare Boot sector and Backup – A comparison of the original and backup boot sectors is performed in this step. Using the DD tool, another 2 boot sector images from the original Image are generated resulting in two image files named, backupbootsector.dd and bootsector.dd. These two files are then analysed using WinHex hex-editor tool, TSK and Autopsy tools.

Step 7: Check the Data Integrity – In this step the integrity of data is verified again for test of congruence. The hashing technique of MD5 tool is adopted to check the data integrity of the two created image files.

Step 8: Extract the ASCII and UNICODE – This step involves extracting the ASCII and UNICODE characters from the binary files available in the disk image. The Strings command tool is used for this and keyword search for matching text or hexadecimal values are recorded on the disk. Through keyword search, even files that contain specific words that could be related to maliciousness are identified.

Step 9: Physical Presentation – In this final step, all the findings from the forensic investigation are documented. It involves presenting the digital evidence through documentation and reporting procedures.

3.8 Boot Sector Analysis of NTFS

3.8.1 NTFS Disk Image

As mentioned in the previous section, the first step to be adopted by a digital forensic investigator is to acquire a duplicate copy of the NTFS disk image before beginning the analysis. This is to ensure that the data on the original devices have not been changed during the analysis. Therefore, it is required to isolate the original infected computer from the disk image in order to extract the evidence that could be found on the electronic storage devices. By conducting investigations on the disk image, any hidden intrusions could be unearthed since the image captures the invisible information as well. The advantages of analysing disk images are that the investigators can: a) preserve the digital crime scene, b) obtain the information in slack space, c) access unallocated space, free space, and used space, d) recover file fragments, hidden or deleted files and directories, e) view the partition structure, and f) get date-stamp and ownership of files and folders.

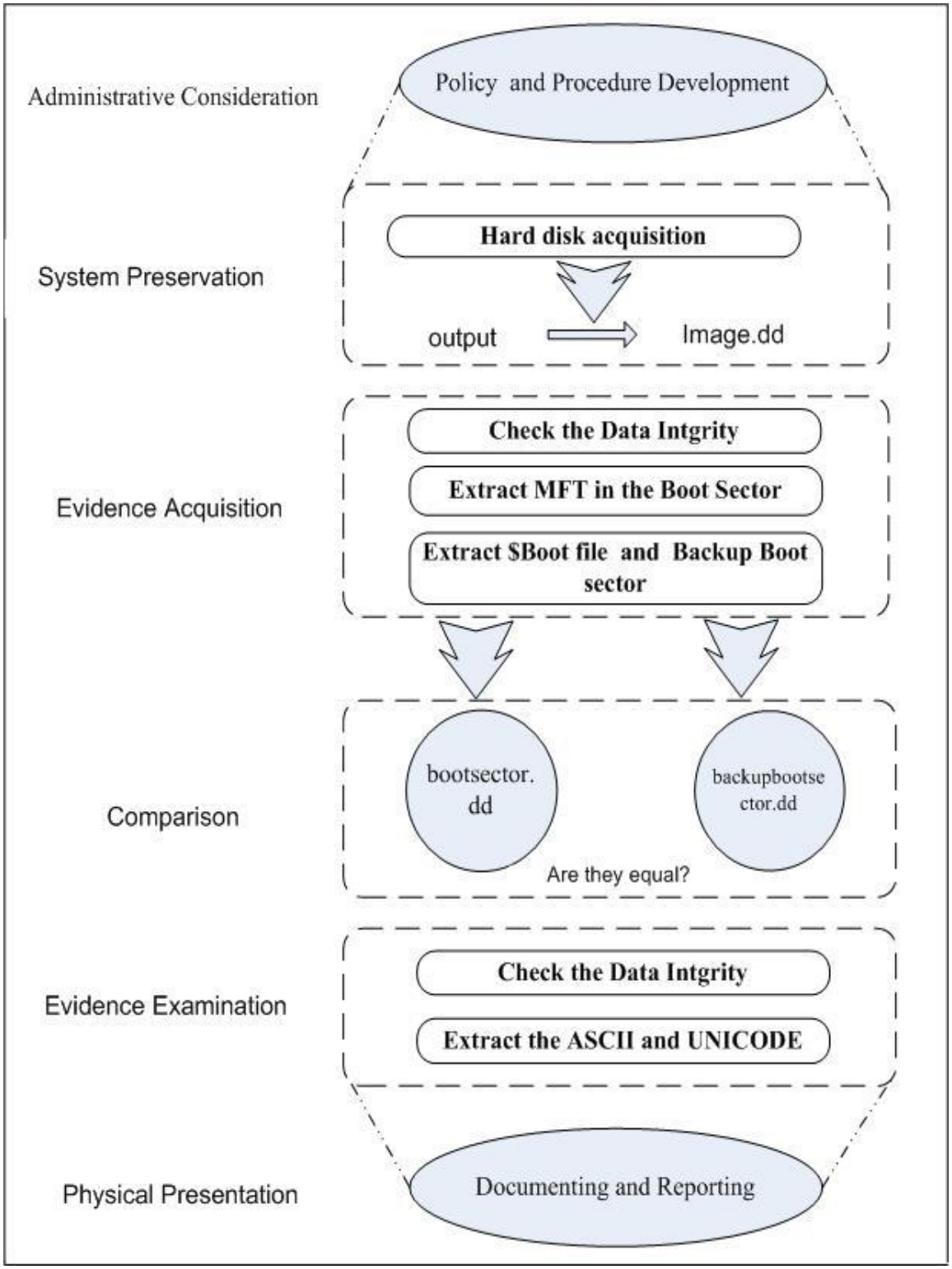


Figure 3.5 Forensic Investigation Steps

3.8.2 Master File Table

To investigate how intrusions take place through data hiding, data deletion and other obfuscations, it is essential to understand the physical characteristics of the Microsoft NTFS file system. Master File Table (MFT) is the core of NTFS since it contains details of every file and folder on the volume and allocates two sectors for every MFT entry (Vacca 2005) . Hence, a good knowledge of the MFT layout structure also facilitates the disk recovery process. Each MFT entry has a fixed size which is 1 KB (at byte offset 64 in the boot sector one could identify the MFT record size). MFT layout has been provided and represents the plan of the NTFS file system using Figure 3.4. The main purpose of NTFS is to facilitate reading and writing of the file attributes and the MFT enables a forensic analyst to examine in some detail the structure and working of the NTFS volume. Therefore, it is important to understand how the attributes are stored in the MFT entry.

3.8.3 Boot Sector Analysis and Results

The boot sector analysis was performed by investigating metadata files used to describe the file system. The steps described in previous section were followed by first creating a NTFS disk image of the test computer using the DD utility for investigating the boot sector. The NTFSINFO tool was run on the disk image. Table 3.3 shows the boot sector of the test device and information about the on-disk structure. Such data structure examination enables the forensic analyst to view the following: MFT information, allocation size, volume size and metadata files. Useful information such as the size of

clusters, sector numbers in the file system, starting cluster address of the MFT, the size of each MFT entry and the serial number given for the file system could be extracted.

From the information gained above, the steps in Figure 3.6 are followed to analyze the boot sector image. As shown in Figure 3.6 and Table 3.4, an analysis of the data structure of this boot sector and the results of the investigation conducted using existing forensic tools is summarized in Table 3.4. From these results, the conclusion is that the existing forensic tools do not check possible infections that could take place in certain hidden data of the boot sector. Hence, the hidden data analysis technique adopted in this research is described in the next section.

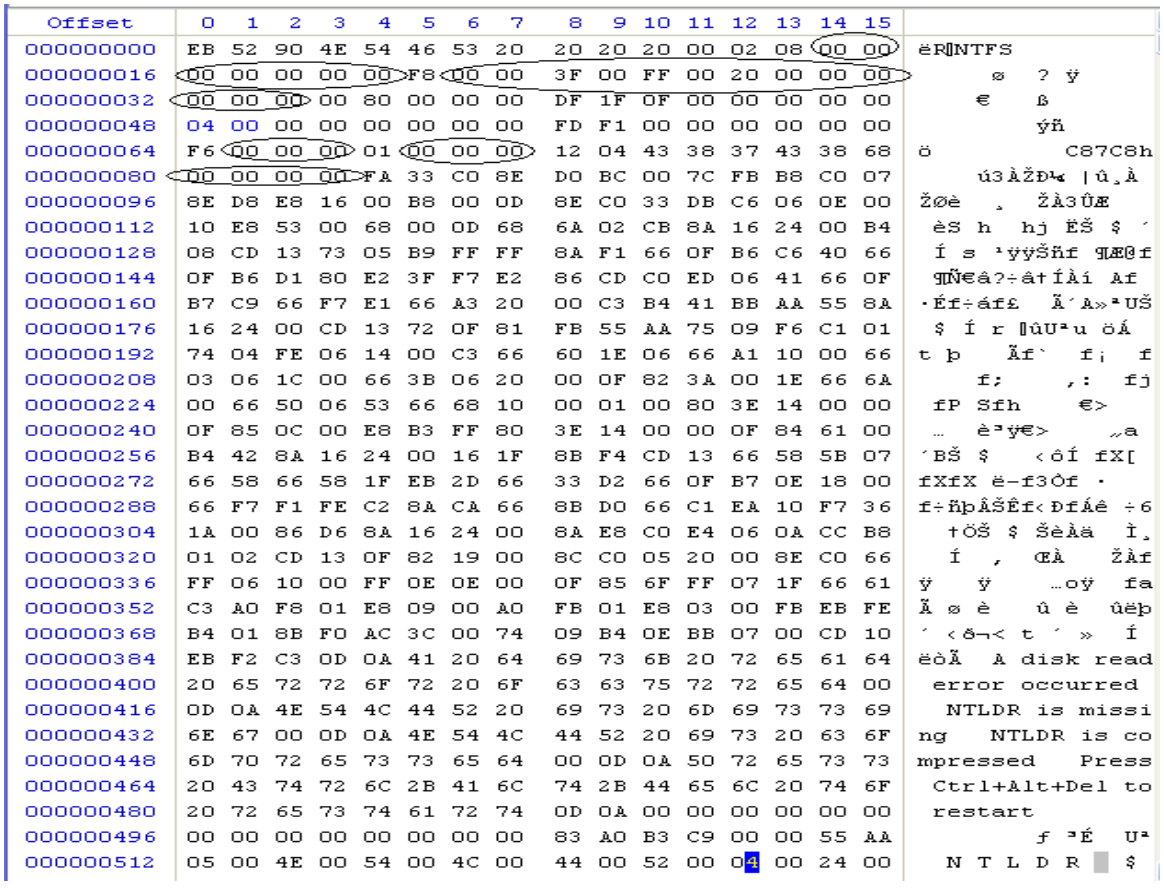


Figure 3.6 Analysis of the Test Boot Sector

Table 3.3 NTFS Information Details

NTFS Information Details

Volume Size

Volume size : 483 MB
Total sectors : 991199
Total clusters : 123899
Free clusters : 106696
Free space : 416 MB (86% of drive)

Allocation Size

Bytes per sector : 512
Bytes per cluster : 4096
Bytes per MFT record : 1024
Clusters per MFT record: 0

MFT Information

MFT size : 0 MB (0% of drive)
MFT start cluster : 41300
MFT zone clusters : 41344 - 56800
MFT zone size : 60 MB (12% of drive)
MFT mirror start : 61949
Meta-Data files

3.9 Hidden Data Analysis and Results

The recent cybercrime trends are to use different obfuscated techniques such as disguising file names, hiding attributes and deleting files to intrude the computer system. Since the Windows operating system does not zero the slack space, it becomes a vehicle to hide data, especially in $Boot file. Hence, in this study, the hidden data is analysed in the $Boot file structure. The $Boot entry is stored in a metadata file at the first cluster in sector 0 of the file system, called $Boot, from where the system boots. It is the only metadata file that has a static location so that it cannot be relocated. Microsoft allocates the first 16 sectors of the file system to $Boot and only half of these sectors contains non-zero values (Huebner *et al.* 2006).

In order to investigate the NTFS file system, one requires possessing substantial knowledge and experience to analyze the data structure and the hidden data. The $Boot metadata file structure is located in MFT entry 7 and contains the boot sector of the file system. It contains information about the size of the volume, clusters and the MFT. The $Boot metadata file structure has four attributes, namely, $STANDARD_INFORMATION, $FILE_NAME, $SECURITY_DESCRIPTION and $DATA. The $STANDARD_INFORMATION attribute contains temporal information such as flags, owner, security ID and the last accessed, written, and created times. The $FILE_NAME attribute contains the file name in UNICODE, the size and temporal information as well. The $SECURITY_DESCRIPTION attribute contains information about the access control and security properties. Finally, the $DATA attribute contains the file contents. These attribute values for the test samples are shown in Table 3.4 as an

illustration. To achieve the invetsigation of NTFS file system, the following TSK command tools were used:

```
Istat  -f  ntfs  c:\image.dd  7
```

By investigating the resulting attribute values, it was observed that the $Boot data structure of the NTFS file system was used to hide data. By analysing the hidden data in the boot sector, one could provide useful information for digital forensics. The size of the data that could be hidden in the boot sector is limited by the number of non-zero that Microsoft allocated in the first 16 sectors of the file system. The data could be hidden in the $Boot metadata files without raising suspicion and without affecting the functionality of the system.

Analysis of the $Boot data structure of the NTFS file system will identify any hidden data. The analyzer should start by making a comparison between the boot sector and the backup boot sector. The image with the boot sector and backup boot sector are supposed to be identical; otherwise there is some data hidden in the $Boot data structure. One method is to check the integrity of the backup boot sector and the boot sector by calculating the MD5 for both of them. A difference in checksum indicates that there is some hidden data. For the experimental investigation, this comparison was done using the following commands on the $Boot image file and the backup boot image:

```
dd if=image.dd bs=512 count=1 skip=61949
of=c:\backupbootsector.dd –MD5sum –verifymd5 –
MD5out=c:\hash1.md5
```

```
dd if=image.dd bs=512 count=1 of=c:\bootsector.dd –MD5sum –
verifymd5 –MD5out=c:\hash2.MD5
```

Table 3.4 Results of $Boot Analysis

$Boot Analysis
MFT Entry Header Values: Entry: 7 Sequence: 7 $LogFile Sequence Number: 0 Allocated File Links: 1 **$STANDARD_INFORMATION Attribute Values:** Flags: Hidden, System Owner ID: 0 Created: Mon Feb 09 12:09:06 2009 File Modified: Mon Feb 09 12:09:06 2009 MFT Modified: Mon Feb 09 12:09:06 2009 Accessed: Mon Feb 09 12:09:06 2009 **$FILE_NAME Attribute Values:** Flags: Hidden, System Name: $Boot Parent MFT Entry: 5 Sequence: 5 Allocated Size: 8192 Actual Size: 8192 Created: Mon Feb 09 12:09:06 2009 File Modified: Mon Feb 09 12:09:06 2009 MFT Modified: Mon Feb 09 12:09:06 2009 Accessed: Mon Feb 09 12:09:06 2009 **Attributes:** Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 48 Type: $FILE_NAME (48-2) Name: N/A Resident size: 76 Type: $SECURITY_DESCRIPTOR (80-3) Name: N/A Resident size: 116 Type: $DATA (128-1) Name: $Data Non-Resident size: 8192 0 1

The main observations from the investigation conducted are that hidden data in the $Boot data structure could not be detected directly by the currently available popular existing forensic tools and laborious manual inspections are required to be performed alongside these tools. Hence, by analysing various existing utilities and tools, the following results have been arrived at:

i. Since NTFS stores all events that take place on a computer system, there is a huge amount of data analysis required while scanning the entire NTFS disk image for forensic purposes. In this empirical study that was focusing on the hidden data of the $Boot file alone, a variety of tools and utilities were required to be adopted along with laborious and knowledge-intensive manual inspections.

ii. The existing forensic tools are not comprehensive and effective in identifying the recent computer threats. Not all computer infections are detected by forensic tools, especially intrusions that are in the form of hidden data in the $Boot file go unchecked.

iii. It is essential to perform manual investigations alongside the existing tools. By adopting a manual introspection of the $Boot file using the three-stage approach of i) hard disk acquisition, ii) evidence searching and iii) analysis of the NTFS file system, it is possible to identify hidden data in the $Boot file.

iv. Intelligent search techniques could be adopted to extract the ASCII and UNICODE characters from binary files in the disk image on either the full file system image or

just the unallocated space, which could speed-up the process of identifying hidden data.

v. One of the main reasons for having varying tools is that Microsoft has different versions of the NTFS file system to be catered for. While Windows XP and Windows Server 2003 use the same NTFS version, Windows Vista uses the NTFS 3.1. The new NTFS 3.1 on Windows 7 has changed the on-disk structure. For example, the location of the volume boot record is at physical sector 2,048. Most of the existing tools do not work with all the different versions of NTFS file system, and hence a comprehensive tool is warranted to cope with these changes.

Table 3.5 Analysis of the Test Boot Sector

Byte Range	Size	Description	Value	Action / Result
0 -- 2	3	Jump to boot code	9458411	If bootable, jump. If non-bootable, used to store error message
3 -- 10	8	OEM Name – System ID	NTFS	
11 -- 12	2	Bytes per sector:	512	
13 -- 13	1	Sectors per cluster	8	
14 -- 15	2	Reserved sectors	0	Unused – Possible Infection
16 -- 20	5	Unused	0	Unused – Possible Infection
21 -- 21	1	Media descriptor	0	
22 -- 23	2	Unused	0	Unused – Possible Infection
24 -- 25	2	Sectors per track	63	No Check – Possible Infection
26 -- 27	2	Number of heads	255	No Check – Possible Infection
28 -- 31	4	Unused	32	No Check – Possible Infection
32 -- 35	4	Unused	0	Unused – Possible Infection
36 -- 39	4	Drive type check	80 00 00 00	For USB thumb drive
40 -- 47	8	Number of sectors in file system (volume)	0.47264 GB	
48 -- 55	8	Starting cluster address of $MFT	4*8=32	
56 -- 63	8	Starting cluster address of MFT Mirror $DATA attribute	619,49	
64 -- 64	1	Size of record - MFT entry	210=1024	
65 -- 67	3	Unused	0	Unused – Possible Infection
68 -- 68	1	Size of index record	01h	
69 -- 71	3	Unused	0	Unused – Possible Infection
72 -- 79	8	Serial number	C87C8h	
80 -- 83	4	Unused	0	Unused – Possible Infection
84 -- 509	426	Boot code	~	
510 --511	2	Boot signature	0xAA55	

Chapter 4 : Anomaly Detection Based on OP-Code

"China and Russia have thousands of well-trained cyberterrorists and we are just sitting ducks".

—*Professor George Ledin,* Sonoma State University

4.1 Overview

Malware that make use of obfuscation of the extended x86 IA-32 operation codes (OP codes) pose a great challenge for malware detectors, as they can easily evade current signature-based, as well as heuristic-based, detection engines. In this chapter, a novel algorithm has been proposed that combines op-code frequency statistics and hybrid wrapper-filter based feature selection technique for constructing a classifier for malware detection. Existing op-code statistical fingerprinting techniques found in the literature are not efficient for analysing large op-code features that are possible due to the numerous obfuscations taking place in reality. The novelty of approach presented in this chapter is that hybridized op-code statistics with a novel wrapper-filter based feature selection technique are used to optimise the process and have achieved the desired efficiency for large datasets. The main contribution of the wrapper-filter based feature selection technique used, is that it is capable of selecting the most important op-codes used in the detection of malware based on the observed patterns of their obfuscation behaviour. The proposed hybrid wrapper-filter approach uses Maximum Relevance-Minimum Redundancy and Artificial Neural Net Input Gain Measurement Approximation (ANNIGMA) and integrates the filter's ranking score with the wrapper-heuristic's score to guide the search process in the wrapper stage, and takes advantage of both the approaches. Also investigated and compared is the malware detection accuracy using

Maximum Relevance (MR), filter ranking heuristics with ANNIGMA, and combined MR-ANNIGMA approaches. Experimental results on large real world malware datasets show that our frequency-statistics based approach achieves high accuracy in all the three cases, with additional efficiency achieved through the MR-ANNIGMA wrapper-filter that arrives at a very compact minimum set of op-codes. Such an approach results in an optimal set of op-codes representative of the innumerable obfuscation patterns adopted by malware attackers would aid in real-time efficiency that is warranted in unknown malware detection.

4.2 Malware Behaviours

Malware affects the secrecy and integrity of data as well as the control flow and functionality of a computer system (Alperovitch 2011). Recent attacks using obfuscated malware (TreadwellZhou & Zhou 2009) have resulted in disruption of services leading towards huge financial and legal implications (Bilar 2007; Lawton 2002; McGraw & Morrisett 2000). Researchers and anti-malware vendors are faced with the challenge of how to detect such zero day attacks (Chouchane & Lakhotia 2006), which is also a major concern for various computer user groups, including home, business and even government users.

Literature surveys on malware detection have shown that there is no single technique that could detect all types of malware. However, there are two techniques commonly used for malware detection, signature-based detection and anomaly-based detection (Birrer *et al.* 2009; Dinaburg *et al.* 2008; Lawton 2002). Anti-Virus engines use malware signatures to detect known malware. The malware signature is a byte sequence that uniquely identifies a specific malware. Typically, a malware detector uses the

malware signature to identify the malware like a fingerprint. Most AV engines are supplied with a database containing information of existing malware to identify maliciousness, by looking for code signatures or byte sequences while scanning the system. A malware detector scans the system for characteristic byte sequences or signatures that match with the one in the database and declares the existence of malware blocking its access to the system. The signature matching process is called signature-based detection and most traditional AV engines use this method. It is a very efficient and effective method to detect known malware (Venkatraman 2009). But, the major drawback is the inability to detect new or unknown malicious code. The signature generation involves manual processing and requires strict code analysis. To overcome signature based methods, polymorphic malware have an in-built polymorphic engine that can generate new variants each time it is executed and a new signature is generated. Therefore, signature based approaches fail to detect such malware. On the other hand, anomaly-based detection uses the knowledge of normal behaviour patterns to decide the maliciousness of a program code. This approach has the ability to detect some zero day attacks. However, it is very difficult to accurately specify the system or program's behaviour and thus these approaches usually are resulting in more false positives than signature based methods.

In this chapter, combines op-code frequency statistics and hybrid wrapper-filter based feature selection technique have been combined in order to construct malware detector. The novelty of the proposed approach is the use of op-code frequency statistics that does not require the signature of malware while detecting the malware in the AV engine. Our contribution also includes the following hitherto unreported in the literature.

1) Statistical differentiation of the op-code frequency is proposed by exploring the op-code relationship with regard to malware and benign programs.

2) A fully automated heuristic method is proposed to disassemble the binary executables and compute op-code frequency statistics

3) A novel hybrid wrapper-filter based feature selection technique is proposed to find the most important op-code for malware. This kind of approach has not been explored in the malware literature yet.

4) A signature-free detection method is proposed to cope with polymorphic transformations and metamorphic obfuscations of malware.

In the proposed approach, a hybrid heuristic of mutual information-based Maximum Relevance and Artificial Neural Net Input Gain Measurement Approximation (ANNIGMA) has been developed which is used along with a novel op-code frequency statistics. The significance of this approach is that it integrates the filter's ranking score with the wrapper-heuristic's score to guide the search process in the wrapper stage that can take the advantages of both approaches and find the most significant op-codes to detect malware.

4.3 Assembly Language and Executable File Format

The language of reversing compiled binary code is the assembly language (Eilam 2005). In this chapter, the focus is on the 'x86' also called the ('IA-32', 386, or the i386-architecture) which is Intel's 32-bit architecture and is the basis for all of Intel's x86 CPU, since the first version of i386 to our day. The focus was on the Intel-32 assembly

language for these experiments, because it is almost exclusively used in every computer and is the most popular processor architecture. The IA-32 architecture and IA-64 are almost same in term of architecture and programming environment, the main difference is that IA-64 bit processors use the prefix/extension to the 80386 instruction set. However, there are popular instructions that most likely could exist in any program either in IA-32 or IA-64 such as moving data, arithmetic or compare operators, conditional branches and function calls. Instead of focusing on the basic instructions ,our study considers all IA-32 instructions and have used the list of instructions from the Intel IA-32 Architecture Software Developer's Manual, Volume 1(Intel 2010a), Volume 2A (Yanfang *et al.* 2010) and Volume 2B (Intel 2010b).

The Win32 Portable Executable (PE) (Christodorescu & Jha 2003) file formats such as (.EXE and .DLL) introduced by Microsoft, which is the standard executable format for all versions of the Windows operating system on all supported processors. As shown in Figure 4.1 PE file has different sections and headers, Windows PE files start with the DOS header which is identified by 'MZ'. The second section is the PE Signature field, which when viewed as ASCII text is **`PE\0\0`**. Third is the **`IMAGE_FILE_HEADER`** containing the most basic information about the file. Fourth, is **`IMAGE_OPTIONAL_HEADER`** that contains the structure of additional information provided by the PE creators, beyond the basic information found in the **`IMAGE_FILE_HEADER`**. Last is the section table that has code sections (**`.text`**), and data sections (**`.data`**). The .text section is the default section for code and the .data section stores writable global variables and also contains the file's Original Entry Point

(OEP) which refers to the execution entry point (where the file execution begins) of a portable executable file. Finally, the **.rdata** section contains read-only data.

The experiment in this chapter have been tested on the PE format files; to compare operation code distributions within malicious and benign files. Automated the inspection of the op-code frequency statistics implemented, and have given a preliminary assessment of its frequency used for detection and differentiation different files of malicious and benign. IDA Pro Dissasember (IDA Pro 2010) has been selected to view and analyse the PE files.

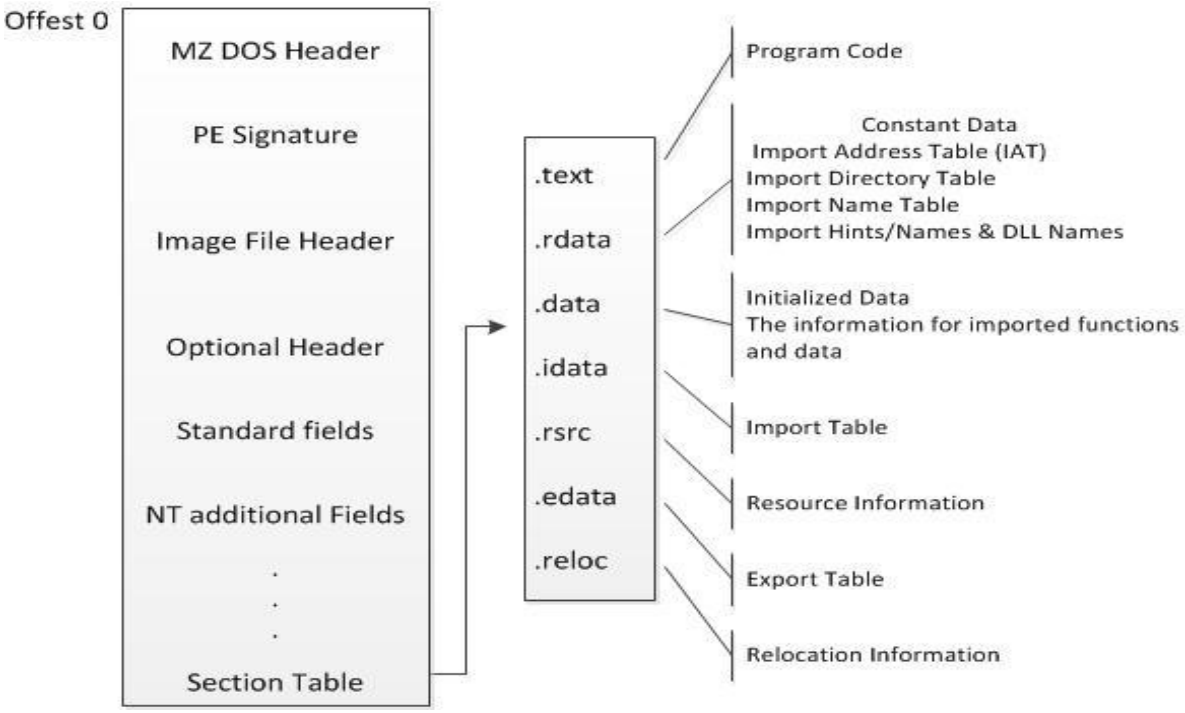
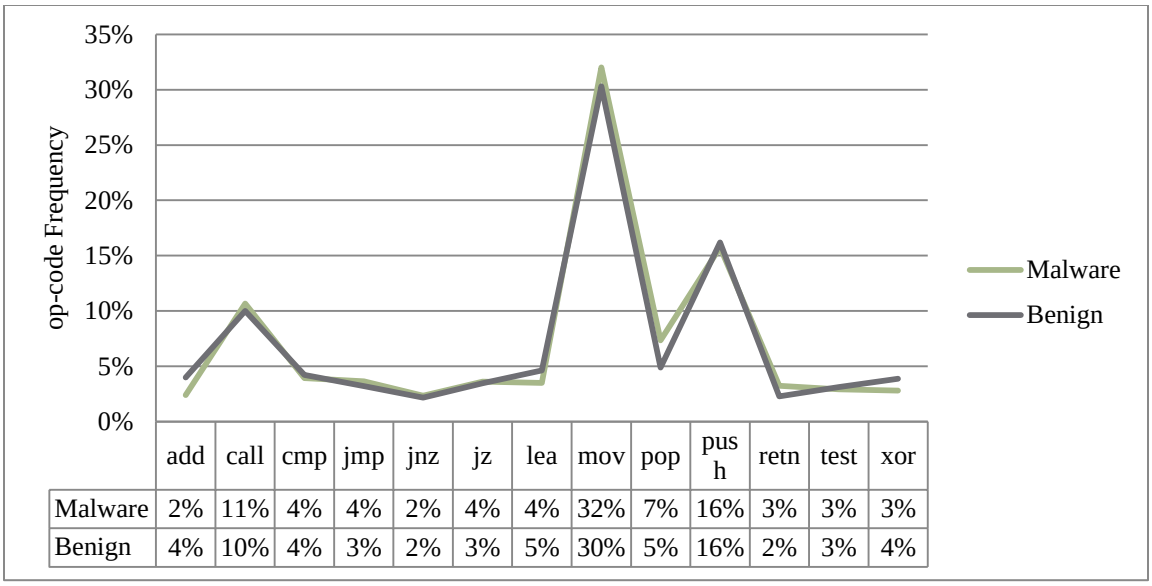


Figure 4.1 The Executable File Format

4.4 Descriptive Analysis of Data

In our dataset for malware and benign files, the aggregate malware dataset yielded roughly about 48,629,512 op-codes and the aggregate benign dataset yielded roughly about 405,942 op-codes. The experiment was run on a total of 590 different op-codes collected from Intel, but for the analysis part the op-codes that have been found in our sample binaries was only just considered which are in total of 80 op-codes. Analysis show that the top 13 listings for both malware and benign are identical (**ADD/ CALL/ CMP/ JMP/ JNZ/ JZ/ LEA/ MOV/ POP/ PUSH/ RETN/ TEST/ XOR**). Many of the new op-codes were not used at all in all our samples such as: Move Data from String to String (**MOVS/ MOVSB/ MOVSW/ MOVSD/ MOVSQ**), Compare String Operand (**CMPS/ CMPSB/ CMPSW/ CMPSD/ CMPSQ**), Load Machine Status Word (LMSW), Load String (**LODS/ LODSB/ LODSW/ LODSD/ LODSQ**) , Repeat String Operation Prefix (**REP/ REPE/ REPZ/ REPNE/ REPNZ**) , Scan String (**SCAS/ SCASB/ SCASW/ SCASD**). Figure 4.2 shows the top 13 listings for both malware and benign and theirs frequent op-codes, the Figure shows that percentage of using the op-codes in malware and clean binaries are almost similar Figures.



	add	call	cmp	jmp	jnz	jz	lea	mov	pop	push	retn	test	xor
Malware	2%	11%	4%	4%	2%	4%	4%	32%	7%	16%	3%	3%	3%
Benign	4%	10%	4%	3%	2%	3%	5%	30%	5%	16%	2%	3%	4%

Figure 4.2 The Most Frequent 13 OP-Codes for Both Malware and Benign

4.5 Proposed OP-Code Detection Methodology

In this chapter a methodology based on op-code analysis is proposed that does not require any previous knowledge of the binary signatures. The new proposed method for finding the fingerprint of executable programs and for detection and differentiation of different files those are malicious and benign. These experiments consider the op-codes based on the extended x86 IA-32 binary assembly instructions. As features approach is to first disassemble the binary executable for OP-Code Frequency statistics (section 4.5.1 and Section 4.5.2) and then to adopt (section 4.6) op-code feature selection algorithms using Maximum Relevance Filter Heuristic and ANNIGMA Wrapper Heuristics (MR-ANNIGMA) for malware detection. By adopting such a methodology obfuscated code such as packing, polymorphism and metamorphism. The proposed signature-free

approach is divided into two main steps which are described in the following sub-sections.

4.5.1 Disassemble Executable for Op-Code Frequency Statistics

In the first step, a fully automated method is developed for extracting op-code frequency statistics from malicious and benign binaries executables. Figure 4.3 shows the system architecture of such an automated process. All samples collected were pre-processed for anomaly testing. In order to translate a program into an equivalent high-level-language program based on the binary content, the most reliable disassembly tool used for static analysis, namely, Interactive Disassembler Pro (IDA Pro 2010) has been chosen which can disassemble all types of non-executable and executable files (such as ELF, EXE, PE, etc.). Also, IDA Pro was selected as a component of the automation process of this research work because it automatically recognizes instruction names of the op-codes for various compilers and can be further extended with our Python programs and compiled plugins, resulting in incredibly powerful implementation with flexible levels of analysis and control. IDA Pro loads a file into memory to analyse the relevant program portion, creating an IDA database whose components are stored in four files: .id0 that contains the content of a B-tree-style database, .id1 that contains flags describing each program byte, .nam that contains index information related to program locations, and .til that is used to store information concerning local type definitions to a given database. IDA Pro generates the IDA database files into a single IDB file (.idb) by disassembling and analysing the binary of the file.

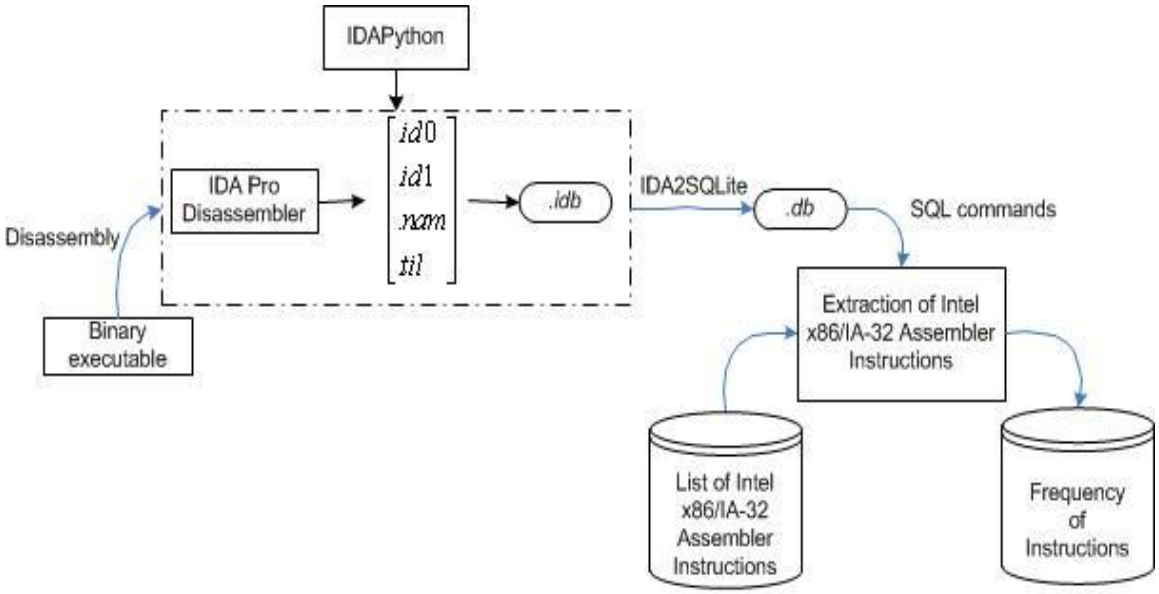


Figure 4.3 Flow Diagrams for OP-Code Frequency Statistics.

IDAPython (IDAPython 2011) has been used, which is an IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro to automate the process, also the C library SQLite (Naiqi *et al.* 2008) has been used, that implements a self-contained SQL database engine with our python program to enable us to convert the binary executable to a database. Therefore, we developed IDA Pro in SQLite name 'IDA2SQLite' plug-in to store the initial analysis results with the extension (.db). Our developed plugin generates eight tables of information, each table contains information about the executable namely Blocks, Functions, Instructions, Names, Maps, Stacks, Segments, and TargetBinaries. Each of these tables contains different information about the binary content. For the analysis of the features, we have run the SQL commands through our Python program to compute the machine op-codes frequency statistics.

4.5.2 Selection of Most Significant OP-Codes

In this step, the frequency statistics of op-codes are used with the wrapper-filter feature selection to construct a signature-free malware detection approach. Significant research works have appeared in the literature on feature selection (Balagani & Phoha 2010; Blum & Langley 1997; Hu *et al.* 2008; John *et al.* 1994; Wolf & Shashua 2005). These can be grouped broadly into three main categories based on the evaluation criteria: 1) the filter model (section 4.6), 2) the wrapper model (Section 4.7), and 3) hybrid models (section 4.9). The filter models are based on the intrinsic characteristics of the data and do not involve the application of an induction algorithm. Diverse filter models have been advanced often involving relevance measures or distance measures as their evaluation criteria (Wang, H. *et al.* 1999). A popular filter approach is Maximum Relevance based on mutual information, which is described in the next section.

4.6 Maximum Relevance Filter Heuristic

Relevant features provide more information about the class variable than irrelevant features. Mutual information based maximum relevance has been proposed as a good heuristic to select salient features within the data mining area (Wang, H. *et al.* 1999). If *S* is a set of features and class variable is c, the maximum relevance can be defined as:

$$Maximum\ Relevance\ (S, c) = \frac{1}{|S|} \sum_{f_i \in S} I\ (F_i; c) \qquad (4.1)$$

$I\ (F_i; c)$ is the mutual information between F_i and class c which is defined as:

$$I\ (F_i; c) = H\ (F_i) - H(F_i \mid c) \qquad (4.2)$$

$H\ (F_i)$ is the entropy of F_i with the probability density function $P\ (f_i)$ where F_i takes discrete values from the set $F = \{f_1, f_2, f_i\}$ then $H\ (F_i)$ is defined as:

$$H\ (F_i) = -\sum_{f_i \in F} P\ (f_i)\ log\ P\ (f_i) \tag{4.3}$$

$H(F_i \mid c)$ in equation (4.2) is the conditional entropy between F_i and c and is defined as:

$$H(F_i \mid c) = -\sum_{f_i \in F} \sum_{C_i \in C} P(f_i, c_i)\ log\ P\ (c_i \mid f_i) \tag{4.4}$$

Where class variable c takes the discrete values from the set $C = \{c_1, c_2, c_i\}$.

In general, filter models are computationally cheap due to their evaluation criteria. However, feature subsets selected by filter may result in poor prediction accuracies, since they are independent from the induction algorithm.

In contrast, wrapper models use a predetermined induction algorithm and use predictive accuracy as the evaluation criteria for the feature selection. Wrapper models face huge computational overhead due to the use of the induction algorithm's performance criteria as their evaluation criteria. In (Hung-Ju *et al.* 2002), (Zhu *et al.* 2007), (Elaiwat *et al.* 2010a) and (Elaiwat *et al.* 2010b) a hybrid of genetic algorithm and filter heuristic was proposed, where GA framework forms the subset generation process, while the filter heuristic improves local search. GA-based approaches face huge computational overheads due to the evaluation of the induction algorithm embedded in the GA fitness function. Some wrapper approaches (Hung-Ju *et al.* 2002) use heuristics

generated from wrapper knowledge over wrapper iteration. A popular wrapper heuristics is Artificial Neural Network Input Gain Measurement Approximation (ANNIGMA).

4.7 ANNIGMA wrapper-heuristic

Artificial Neural Network Input Gain Measurement Approximation (ANNIGMA) is a weight analysis based wrapper heuristic that ranks features by relevance based on the weight associated with feature in a Neural Network based wrapper approach (Hung-Ju *et al.* 2002). Features that are irrelevant or redundant will produce more error than relevant features. Therefore, during training, weights of noisy features are controlled in such a way that they contribute to the output as little as possible. ANNIGMA is based on the above strategy of the training algorithm. As shown in Figure 4.4, for a two layer Neural Network, if i, j, k are the input, hidden and output layers and *Q* is a logistic activation function (4.1) of the first layer and second layer has a linear function, then the output of the network Q_k is as (4.2). Here A_i are the input feature and *W* are the weights between network layers.

$$Q(x) = \frac{1}{1 + exp(-x)} \tag{4.5}$$

$$Q_k = \sum_j Q\left[\left[\sum_i A_i \times W_{ij}\right]\right] \times W_{jk} \tag{4.6}$$

Then local gain is defined as:

$$LG_{ik} = \frac{\Delta O_k}{\Delta A_i} = \sum_j |W_{ij} \times W_{jk}| \tag{4.7}$$

Then ANNIGMA score is the local gain normalized on a unity scale as equation (4.8) (Hung-Ju *et al.* 2002):

$$ANNIGMA_{ik} = \frac{LG_{ik}}{max(LG_k)} \tag{4.8}$$

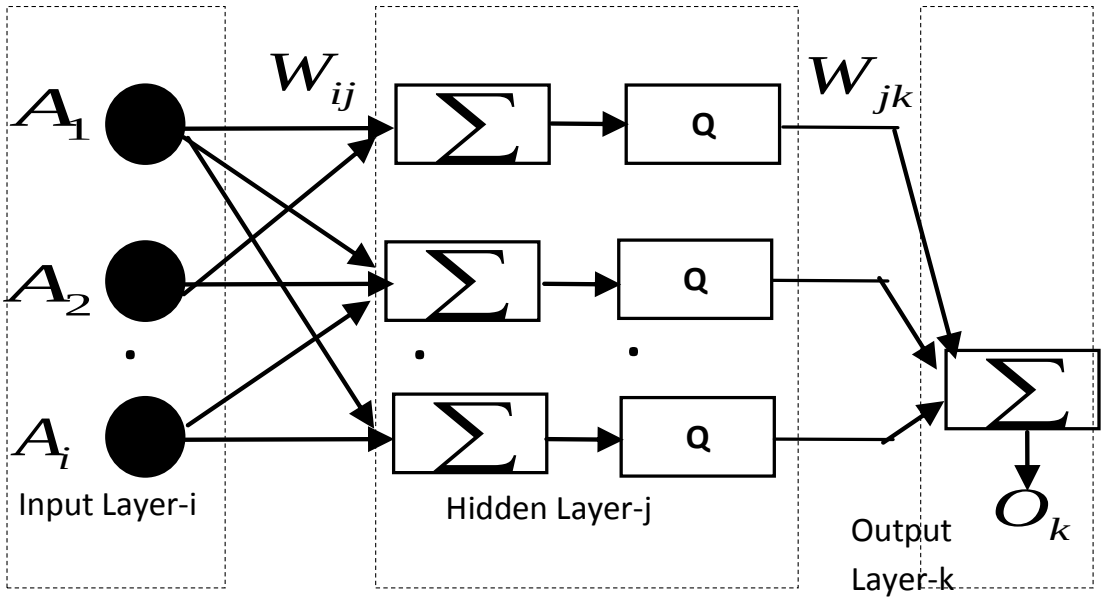


Figure 4.4 A Single Hidden Layer Multi-Layer Perceptron Neural Network in Wrapper Approach

4.8 OP-Code Selection

The proposed op-code-selection and malware detection algorithm in this chapter is a hybrid of the Wrapper-Filter approach using Maximum Relevance-based Filter Heuristic and the Artificial Neural Network Input Gain Measurement Approximation (ANNIGMA) Wrapper Heuristics. Standard filter approaches can extract knowledge of the intrinsic characteristics from real data. However, filter approaches do not use any performance criteria based on predictive accuracies. This does not guarantee that selected feature

subset will be able to do better in the classification tasks. In contrast, the wrapper approach uses a predetermined induction algorithm and different search strategies to find the best feature subset. Use of predictive-accuracy based evaluation criteria in the wrapper ensures good performance from the selected feature subset. However, repeated execution of the induction algorithm (in the worst case exponential search space) in the search process incurs a high computational cost in the wrapper approach. Earlier research shows that a hybrid of wrapper-filter heuristic significantly improves the performances while applying in data mining applications (Huda *et al.* 2010).

In this chapter, a hybrid approach is proposed that introduces the filter heuristic in the wrapper stage and take advantage of both approaches which is able to find more significant op-codes than either wrapper or filter alone to find the Malware. The idea behind this approach can be explained by the Venn-diagram as shown in Figure 4.5. If the two feature subsets ACBF and ADBE are separately ordered/ ranked according to their score, then common higher ranked feature subset (ACBD) is the strongly recommended most significant feature subset by both feature selection algorithms. If the scores of both algorithms are normalized on the same scale and combined, then feature subsets with higher combined scores provide the common higher ranked feature subset from both algorithms. A Backward Elimination (BE) search strategies based on the combined score along with the wrapper evaluation criteria can find the most significant features. Performance of the combined score may be affected due to the performance of the incorporated filter for a particular wrapper approach in the hybrid. However, different filter approaches can be combined to find a suitable hybrid for a particular wrapper heuristic and vice-versa. In this chapter, we have combined mutual information based

Filter-Maximum Relevance (MR) with Artificial Neural Network Input Gain Measurement Approximation (ANNIGMA) based wrapper. Here, we have focused on a Neural network based wrapper and different filter heuristics. We will use other wrapper approaches in a future work. The following sub-sections describe the proposed hybrid algorithm.

The proposed hybrid approach avoid the computational overhead of hybrid GA-based approaches takes advantage of both filter and wrapper heuristics which are absent in the traditional GA-based hybrid approaches (Hung-Ju *et al.* 2002; Kohavi & John 1997). This type of hybrid approach is a new concept and has not been explored yet in the malware detection literature. The different sub-components in the hybrid approach have been shown in Figure 4.6 and main steps are described in the follow sections.

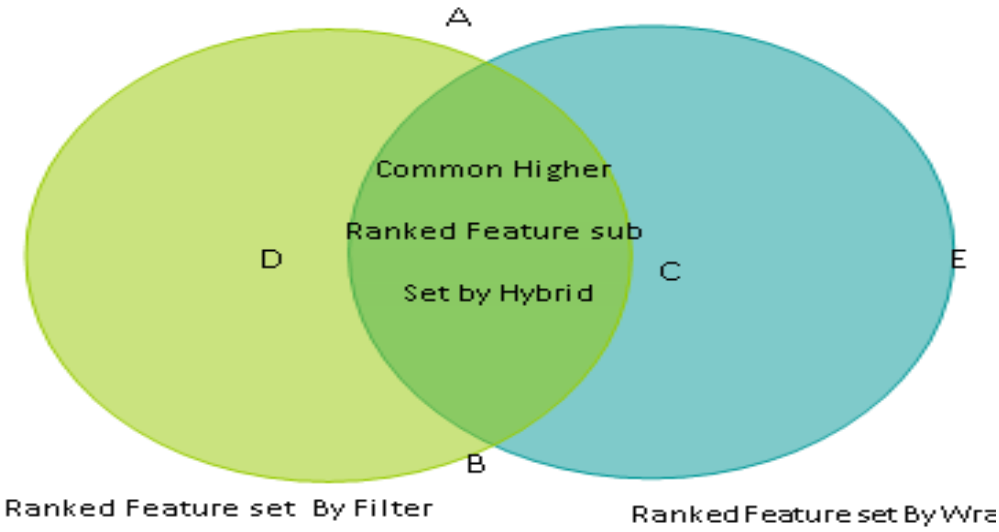


Figure 4.5 Venn Diagram for Hybrid Algorithm

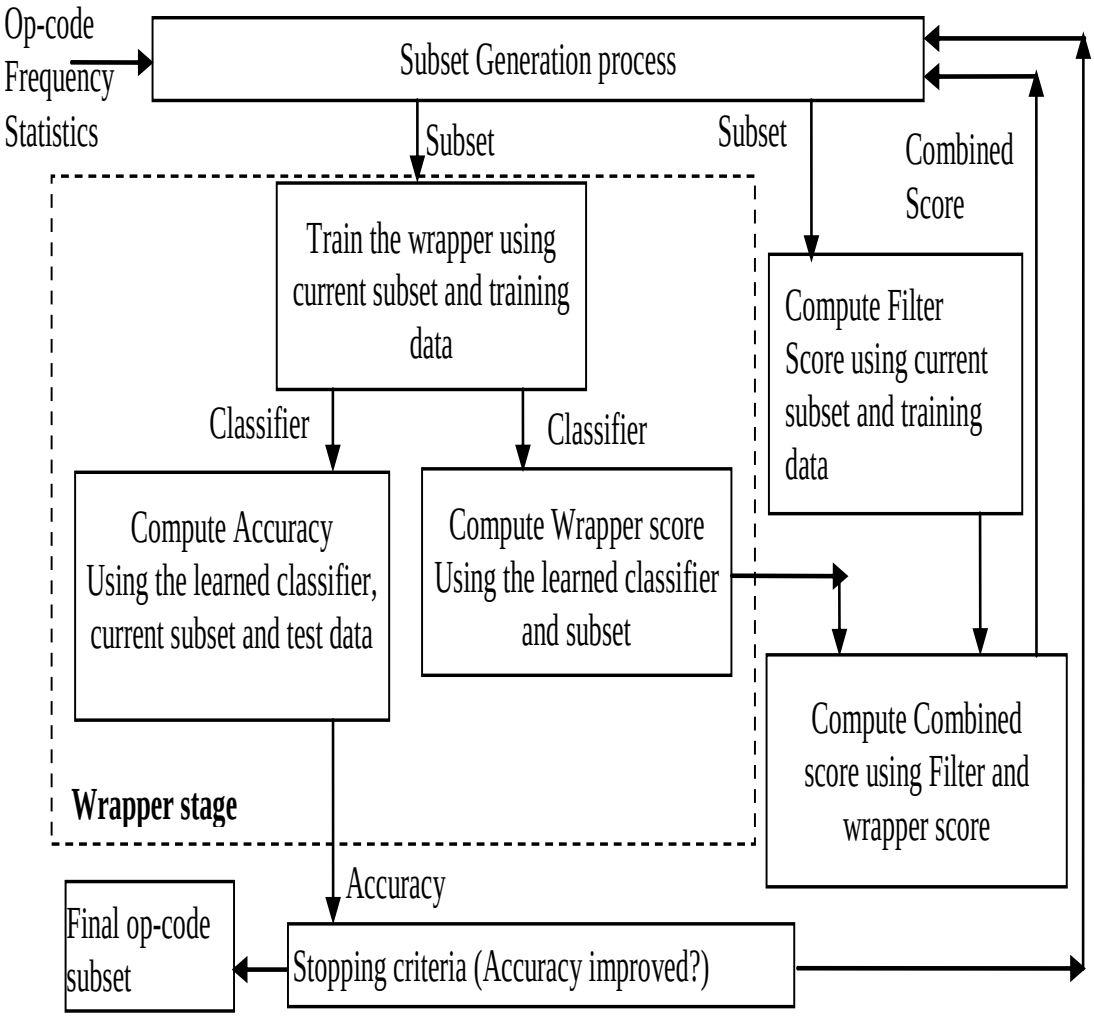


Figure 4.6 Framework for Hybrid of Wrapper and Filter Feature Selection

Algorithm 1: Procedure Hybrid Wrapper-Filter approach for Malware detection

Input: $D(F_1, F_2, ...F_m)$ // Training data with m features

Output: S_{BEST} //an optimal subset of features

Begin

1. Let S=whole set of m features $F_1, F_2, ...F_m$

2. S_0 =Initial set of feature which records all generated subsets with accuracy

// Apply a Backward Elimination (BE) search strategy

3. for N = 1 to m-1

4. Current set of feature $S_{current}$ =S

5. Compute Filter score by (11)

6. for fold=1 to n

7. Train the network with $S_{current}$

8. Compute ANNGMA of all features

9. Compute Accuracy

10. endfor

11. Compute average accuracy of all folds for $S_{current}$

12. Compute average ANNIGMA of $S_{current}$ by (11)

13. Compute combined score for every feature in $S_{current}$ by (11-13) for both hybrids

14. Rank the features in $S_{current}$ using the combined score in descending order

15. $S_0 = S_0 \cup S_{current}$

16. Update the current feature set $S_{current}$ by removing the feature with lowest score

17. endfor

18. S_{BEST} = Find the subset from S_0 with the highest accuracy.

19. return S_{BEST}

End

4.9 Hybrid Models

The computation of combined score Hybrid of Maximum Relevance and ANNIGMA based signature-free for malware detection technique uses Artificial Neural Network as

the induction algorithm in the wrapper. An n-fold cross-validation approach has been used in MR-ANNIGMA to train the wrapper. In each fold we compute the ANNIGMA score for every feature. Then after training of all folds, the ANNIGMA score is averaged as (4.9):

$$ANNIGMA\ (F_i)_{Avg} = \left(\frac{1}{n}\right) \sum_{j=0}^{n} ANNIGMA\ (F_i)_n \quad (4.9)$$

While computing the combined score in the proposed ANNIGMA, the relevance of a feature in the current subset is computed from the individual score which is scaled to the maximum individual relevance of the subset. The, relevance of a feature in a subset within the hybrid approach is defined as given in equation (4.10)

$$Relevance\ (F_i) = \frac{I(F_i;c)}{Max\ (f_i \in S)\ I(F_i;c)} \quad (4.10)$$

The combined score of the filter's heuristic and the wrapper's heuristic in the proposed MR-ANNIGMA is computed as in equation (4.11)

$$Combined\ Score\ (MR - ANNIGMA)$$

$$= \frac{I(F_i; c)}{Max\ (f_i \in S)\ I(F_i; c)} + ANNIGMA\ (F_i)_{Avg.} \quad (4.11)$$

The detail of algorithm of this hybrid approach is described in Algorithm 1.

4.9.1 Search Strategies

MR-ANNIGMA uses a Backward Elimination (BE) search strategy to generate a subset of op-codes. Initially, it starts with the full op-code set. Subset generation in BE is guided

by the wrapper-filter hybrid heuristic score. The combined score computation follows the steps of sub-sections. When the number of op-codes in BE process is significantly reduced compared to the total op-codes, the filter score component is weighted less than the wrapper score as given in equation (4.12)

$$Combined\ Score\ (MR-ANNIGMA) = u * \frac{I(F_i;c)}{Max\ (f_i \in S)\ I(F_i;c)} + v * ANNIGMA\ (F_i)_{Avg.} \quad (4.12)$$

Where $\mathbf{1} \leq \boldsymbol{u}, \boldsymbol{v} \geq \mathbf{0}$

4.9.2 Wrapper Step in MR-ANNIGMA

MR-ANNIGMA uses a single hidden layer Multi-Layer Perceptron (MLP) Network (Figure 4.4) in the wrapper stage. An n-fold cross validation approach has been applied in the training of the network. The evaluation criterion of op-code subset is based on the average prediction accuracy over n-fold of the wrapper (MLP network). In Algorithm 1, steps 1 to 5 compute the filter score of the current feature subset. Step 6 to 11 compute the average accuracy over n-folds and compute the wrapper score for the current subset of op-codes. Step 12 to step 14 computes the hybrid scores and the op-codes are ranked based on their combined score. Step-15 to step-16 would then generate a new subset based on the op-code ranking and would keep a record of evaluated op-code subsets with their accuracy. The BE processes in MR-ANNIGMA would update the MR and ANNIGMA and the combined score in every iteration. The combined score then guides the subset generation. The BE continues until a single op-code remains in the current

subset. The subset with the highest accuracies or close to the highest accuracies with fewer op-code set is then chosen as the final op-code subset for the detector.

4.10 Discussion on Experimental Results

The proposed MR-ANNIGMA based signature-free approach has been tested on a large set of real and recent malware samples for detecting more unknown malware. We have two types of datasets, namely malware, and executable benign files or 'good wares'. For the malware dataset, we have collected the infected files from honeypots, honeynet project and other sources, and for benign datasets, we have considered different good files such as, application software (Educational software, Mathematical software, Image editing, Spreadsheet, Word processing, Decision making software, Internet Browser, Email and system software and Programming languages software, and many others). 1,474 executable files have been used in which 450 are benign files and 1,024 malware samples that have been uniquely named according to their MD5 value. Overall, the op-code set consists of 97 op-codes, and the list was gathered from Intel, based on the extended x86 IA-32 binary assembly instructions. A single hidden layer neural network with 22 hidden nodes and (tansig-- hidden layer function and purelin—output layer function) is used as Figure 4.5.

Experimental results have been described in Table 4.1 and Table 4.2 provide the score for filter approach MR, wrapper approach ANNIGMA, and combination of filter approach MR and wrapper approaches ANNIGMA (MR- ANNIGMA). The backward elimination (BE) process starts with all 97 op-codes and accuracies of different iterations of the BE process for all three algorithms have been given in Table 4.2. With all 97 op-codes, ANNIGMA achieves accuracies of 96.029 %, MR achieves accuracies of 96.002% and

MR-ANNIGMA achieves 95.820%. The op-codes are sorted according to their score. In the same Table 4.1 hybrid score of MR-ANNIGMA has been provided. In the second iteration, the op-code with lowest score is discarded and then the subset is evaluated.

The BE process continues, while total op-codes 28, in Figure 4.7, the hybrid re-computes all op-code scores resulting in op-code 6 attaining the lowest for ANNIGMA, op-code 19 attaining the lowest for MR and op-code 74 attaining the lowest for combined score. Therefore in this iteration, op-code 74 is eliminated and MR-ANNIGMA achieves an accuracy of 97.203%. In the next cycle, MR-ANNIGMA eliminates op-code-58 due to its lowest combined score in Figure 4.7 and accuracy of hybrid increases to 97.434%. In Figure 4.9 when total op-code is 17, MR-ANNIGMA eliminates op-code 22 and in Figure 4.10 op-codes 90 is eliminated at total-16 op-codes (due their lowest combined score) where MR-ANNIGMA achieves accuracy of 97.434%. In the next cycle, BE process eliminates op-code-97 for lowest combined score in Figure 4.11 at total-15 op-codes. The BE process continues and ANNIGMA achieves (96.877%) accuracies with 25 op-code set and accuracies of ANNIGMA drops to 68% with one op-code for ANNIGMA. MR achieves 97.475% accuracies with 27 op-codes as shown in Figure 4.8 and 97.405% with 26 op-codes. MR-ANNIGMA achieves an accuracy of 97.573% at total op-code 24 and an accuracy of 97.597% at total op-code 23. Accuracies for MR-ANNIGMA decreases to 93.944% for one op-code. MR-ANNIGMA achieves (97.529%) accuracies as given in Table 4.2 with smallest set op-codes (15 only) which is the closest to the accuracy level at the op-code set 23 with highest accuracy 97.597%. Therefore, op-code set 15 has been considered as the final and most significant op-code set for MR-ANNIGMA. The accuracy results in the BE process as shows in the same table (Table

4.2) that hybrid approach proposed is achieves the highest accuracies with smallest op-codes sets for detection of Malware.

The final op-codes sets from all three algorithms (ANNIGMA, MR and MR-ANNIGMA) have been used in a 10-fold cross validation set. The class discriminative performance of the most significant op-code sets form all three algorithms has been tested by varying the NN's threshold values of the output node in the cross-validation. Then average sensitivity and specificity over 10-fold have been used to produce a Receiver Operating Characteristics (ROC) curve for each algorithm which have been presented in Figure 4.12. The ROC curve of MR-ANNIGMA achieves the highest sensitivity with the highest specificity in all three algorithms. This demonstrates the efficacy of the proposed hybrid algorithm MR-ANNIGMA in Malware detection.

Our experimental results show that the proposed wrapper-filter based approach finds a small op-code set for malware detection while maintaining very high accuracy and sensitivity levels with high specificity. The proposed approach does not need any signature to detect malware. However, further investigation could be made using other wrapper approach. One of the limitations of the proposed approach is that it is supervised and needs to re-train, which may be difficult for end user. However, this limitation could be avoided by using a rule generation step along with the result of the proposed approach. In such a case, the users are required to update only the rules. This would be explored in future research along with more investigations with regard to feature selection using other wrapper approaches.

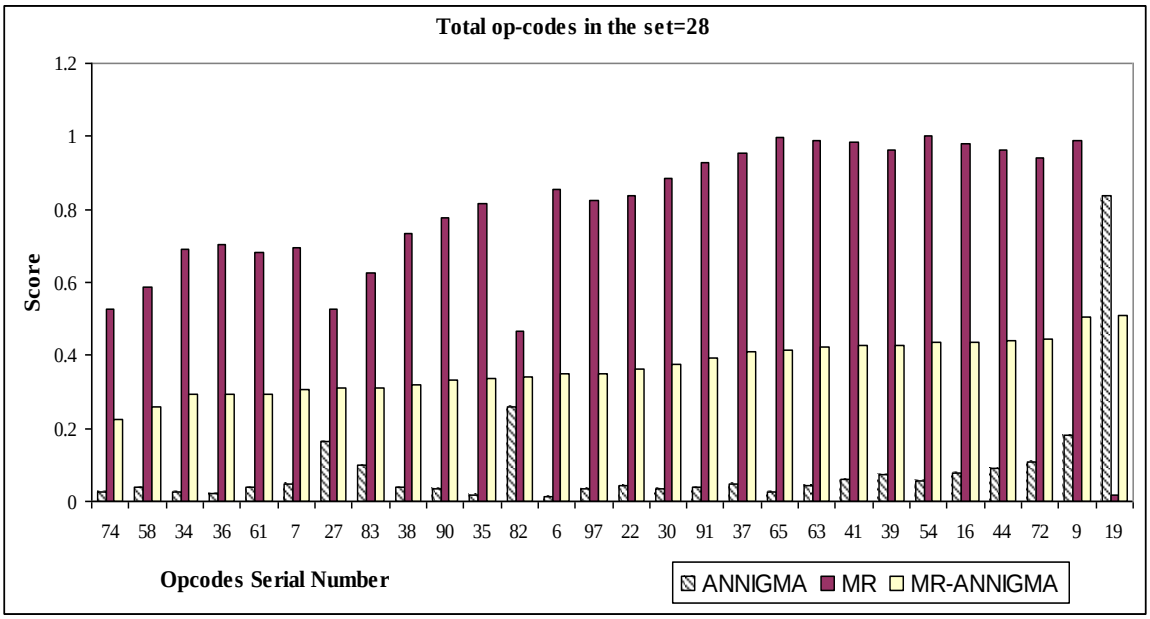


Figure 4.7 Score of OP-Codes When Total Attributes=28

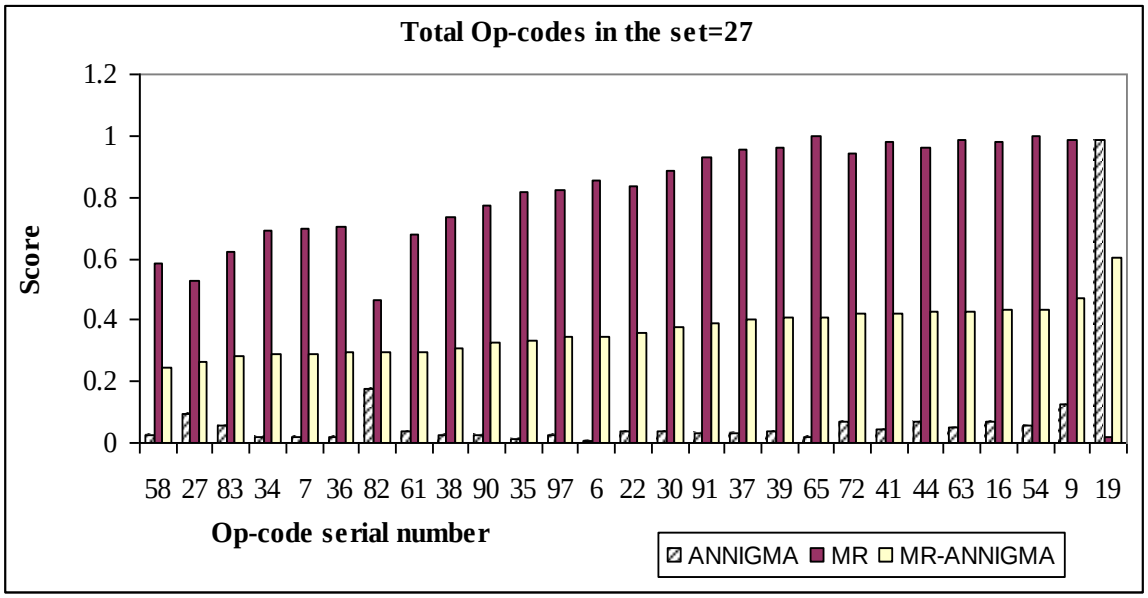


Figure 4.8 Score of OP-Codes When Total Attributes=27

Table 4.1 Heuristics Score for MR, ANNIGMA and MR-ANNIGMA

OP-Code	ANNIGMA	MR	MR-ANNIGMA	OP-Code	ANNIGMA	MR	MR-ANNIGMA	OP-Code	ANNIGMA	MR	MR-ANNIGMA
AAA	0.093	0.011	0.104	**JA**	0.010	0.691	0.701	**RCL**	0.030	0.521	0.551
AAD	0.288	0.010	0.298	**JB**	0.009	0.814	0.823	**RCR**	0.011	0.091	0.102
AAM	0.024	0.021	0.045	**JBE**	0.014	0.703	0.717	**REPXX**	0.003	0.000	0.003
AAS	0.076	0.015	0.092	**JMP**	0.024	0.954	0.977	**RET**	0.003	0.000	0.003
ADC	0.011	0.474	0.485	**JNB**	0.017	0.733	0.750	**RETF**	0.108	0.185	0.293
ADD	0.011	0.854	0.865	**JNZ**	0.049	0.962	1.011	**RETN**	0.043	0.943	0.986
AND	0.008	0.696	0.704	**JXX**	0.002	0.000	0.002	**ROL**	0.046	0.429	0.475
ARPL	0.175	0.007	0.182	**JZ**	0.009	0.982	0.992	**ROR**	0.080	0.526	0.606
CALL	0.155	0.987	1.142	**LAR**	0.051	0.003	0.054	**SAHF**	0.045	0.260	0.305
CBW	0.328	0.018	0.346	**LDC**	0.002	0.000	0.002	**SAL**	0.045	0.007	0.052
CLC	0.028	0.025	0.053	**LEA**	0.093	0.962	1.056	**SAR**	0.009	0.492	0.501
CLD	0.013	0.445	0.458	**LGDT**	0.013	0.001	0.014	**SBB**	0.009	0.254	0.264
CLI	0.101	0.026	0.128	**LIDT**	0.004	0.001	0.005	**SCASB**	0.003	0.000	0.003
CLTS	0.005	0.000	0.005	**LMSW**	0.002	0.000	0.002	**SCASW**	0.002	0.000	0.002
CMC	0.041	0.117	0.158	**LODSB**	0.003	0.000	0.003	**SGDT**	0.003	0.000	0.003

CMP	0.021	0.978	0.999	**LODSW**	0.002	0.000	0.002	**SHL**	0.098	0.467	0.565
CMPSB	0.003	0.000	0.003	**LOOP**	0.026	0.227	0.253	**SHR**	0.053	0.624	0.677
CMPSW	0.003	0.000	0.003	**LOOPX**	0.002	0.000	0.002	**SIDT**	0.016	0.002	0.017
CWD	0.279	0.018	0.297	**LSL**	0.075	0.004	0.079	**STC**	0.016	0.040	0.057
DAA	0.113	0.015	0.128	**LTR**	0.007	0.000	0.007	**STD**	0.008	0.410	0.418
DAC	0.105	0.008	0.113	**MOV**	0.010	1.000	1.010	**STI**	0.029	0.025	0.054
DEC	0.017	0.837	0.854	**MOVSB**	0.003	0.000	0.003	**STOSB**	0.003	0.000	0.003
DIV	0.015	0.553	0.568	**MOVSW**	0.002	0.000	0.002	**STR**	0.009	0.000	0.010
ESC	0.003	0.000	0.003	**MUL**	0.015	0.110	0.124	**SUB**	0.020	0.775	0.794
FLDCW	0.029	0.302	0.331	**NEG**	0.019	0.585	0.604	**TEST**	0.012	0.927	0.939
HLT	0.788	0.005	0.793	**NOP**	0.066	0.111	0.177	**VERR**	0.012	0.002	0.014
IDIV	0.080	0.526	0.606	**NOT**	0.006	0.308	0.314	**VERW**	0.003	0.000	0.003
IMUL	0.027	0.364	0.391	**OR**	0.016	0.681	0.697	**WAIT**	0.098	0.295	0.393
IN	0.170	0.021	0.190	**OUT**	0.269	0.020	0.289	**XCHG**	0.007	0.375	0.382
INC	0.028	0.887	0.914	**POP**	0.016	0.989	1.005	**XLAT**	0.076	0.017	0.093
INT	0.038	0.042	0.080	**POPF**	0.020	0.030	0.050	**XOR**	0.026	0.824	0.850
INTO	0.284	0.006	0.290	**PUSH**	0.023	0.999	1.022				
IRET	0.087	0.015	0.101	**PUSHF**	0.031	0.037	0.068				

Table 4.2 Accuracies at Different Iteration for MR, ANNIGMA and Proposed Hybrid Approach MR- ANNIGMA

OP-Code	ANNIGMA	MR	MR-ANNIGMA	OP-Code	ANNIGMA	MR	MR-ANNIGMA	OP-Code	ANNIGMA	MR	MR-ANNIGMA
AAA	68.83	94.71	93.94	**JA**	96.35	97.20	97.31	**RCL**	96.33	96.38	96.71
AAD	68.42	94.85	94.28	**JB**	96.00	96.88	97.05	**RCR**	96.10	96.55	96.70
AAM	95.30	95.53	94.90	**JBE**	96.42	96.84	97.11	**REPXX**	96.52	96.43	96.77
AAS	95.34	95.34	95.25	**JMP**	96.50	97.39	97.26	**RET**	96.40	96.38	96.97
ADC	95.47	96.51	95.25	**JNB**	96.42	97.16	96.89	**RETF**	96.19	95.93	96.20
ADD	95.41	96.58	95.71	**JNZ**	96.29	97.07	97.03	**RETN**	96.32	96.38	96.19
AND	95.33	96.50	95.57	**JXX**	96.13	96.92	96.81	**ROL**	96.46	96.46	96.27
ARPL	95.28	96.63	95.71	**JZ**	96.31	96.77	96.82	**ROR**	96.38	96.74	96.40
CALL	95.29	96.80	96.33	**LAR**	96.04	96.51	96.71	**SAHF**	96.67	96.00	96.28
CBW	95.18	96.55	96.39	**LDC**	96.04	96.99	96.92	**SAL**	96.66	96.43	96.21
CLC	95.28	96.84	96.35	**LEA**	95.90	97.18	97.01	**SAR**	96.67	96.73	96.59
CLD	95.42	96.73	96.32	**LGDT**	96.28	96.65	96.69	**SBB**	96.35	96.13	96.04
CLI	95.40	96.39	96.92	**LIDT**	96.42	97.12	96.66	**SCASB**	96.13	96.25	96.29
CLTS	95.11	96.70	97.30	**LMSW**	96.38	96.52	96.92	**SCASW**	96.17	95.85	96.42
CMC	96.00	96.70	97.53	**LODSB**	96.25	96.88	96.97	**SGDT**	96.61	96.35	96.48

CMP	96.12	96.76	97.43	**LODSW**	96.36	96.76	96.61	**SHL**	96.48	96.14	96.61
CMPSB	96.28	96.92	97.07	**LOOP**	96.61	96.77	96.77	**SHR**	96.47	96.54	96.52
CMPSW	96.25	96.96	96.89	**LOOPX**	96.42	96.84	96.55	**SIDT**	96.59	96.42	96.47
CWD	96.43	96.31	97.37	**LSL**	96.85	96.78	96.63	**STC**	96.14	96.33	96.27
DAA	96.50	96.97	97.23	**LTR**	96.61	96.28	96.77	**STD**	96.43	96.16	96.48
DAC	96.55	96.70	97.07	**MOV**	96.52	96.55	96.25	**STI**	96.13	95.89	96.54
DEC	96.05	96.09	97.41	**MOVSB**	96.78	96.32	97.08	**STOSB**	96.35	96.10	96.16
DIV	96.77	97.00	97.60	**MOVSW**	96.39	96.51	96.73	**STR**	96.54	96.39	96.21
ESC	96.78	97.14	97.58	**MUL**	96.54	96.36	96.99	**SUB**	96.40	96.19	96.40
FLDCW	96.88	97.37	97.31	**NEG**	96.48	96.88	96.38	**TEST**	96.16	96.42	96.50
HLT	96.80	97.41	97.54	**NOP**	96.62	96.61	96.93	**VERR**	96.57	96.48	96.33
IDIV	96.12	97.48	97.43	**NOT**	96.71	96.52	96.61	**VERW**	96.09	96.36	95.89
IMUL	96.33	97.27	97.20	**OR**	96.36	96.85	96.86	**WAIT**	96.25	96.50	96.69
IN	96.80	97.61	97.16	**OUT**	96.47	96.65	96.17	**XCHG**	96.15	96.08	96.32
INC	96.14	97.31	97.22	**POP**	96.33	96.82	96.58	**XLAT**	95.86	96.05	95.99
INT	96.59	97.41	97.38	**POPF**	96.31	96.66	96.89	**XOR**	96.03	96.00	95.82
INTO	96.63	97.19	97.27	**PUSH**	96.54	96.66	96.51				
IRET	96.51	97.48	97.61	**PUSHF**	96.48	96.78	96.65				

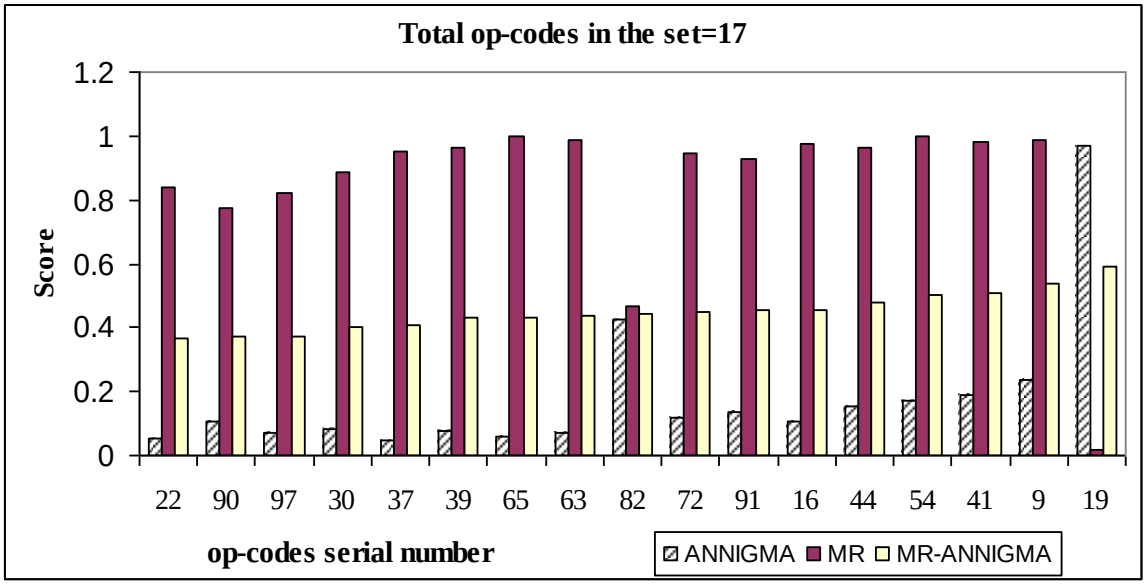


Figure 4.9 Score of OP-Codes When Total Attributes=17

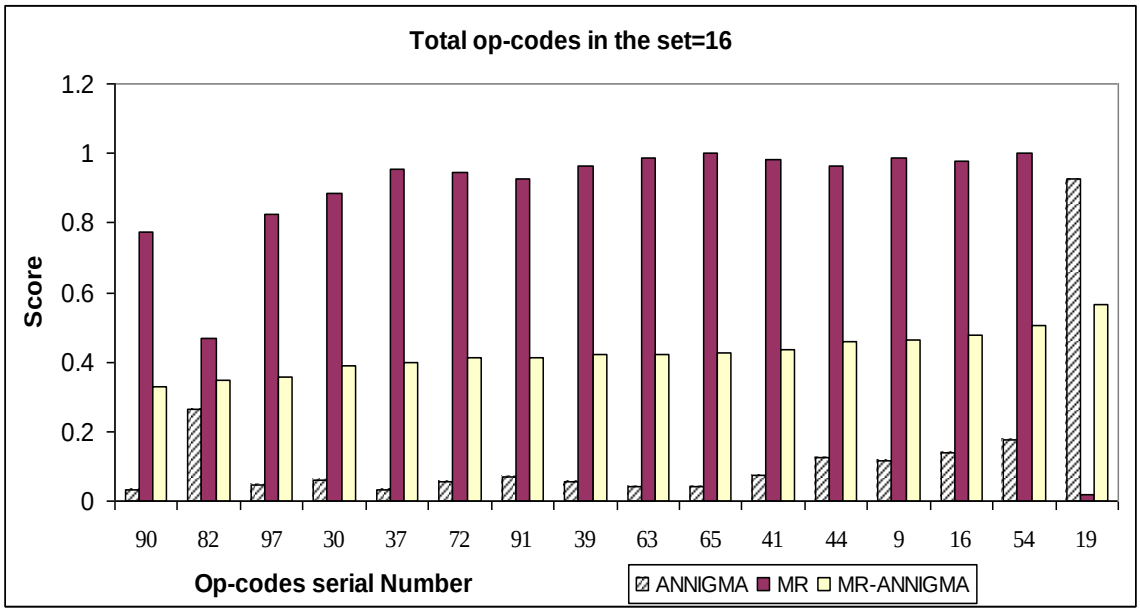


Figure 4.10 Score of OP-Codes When Total Attributes=16

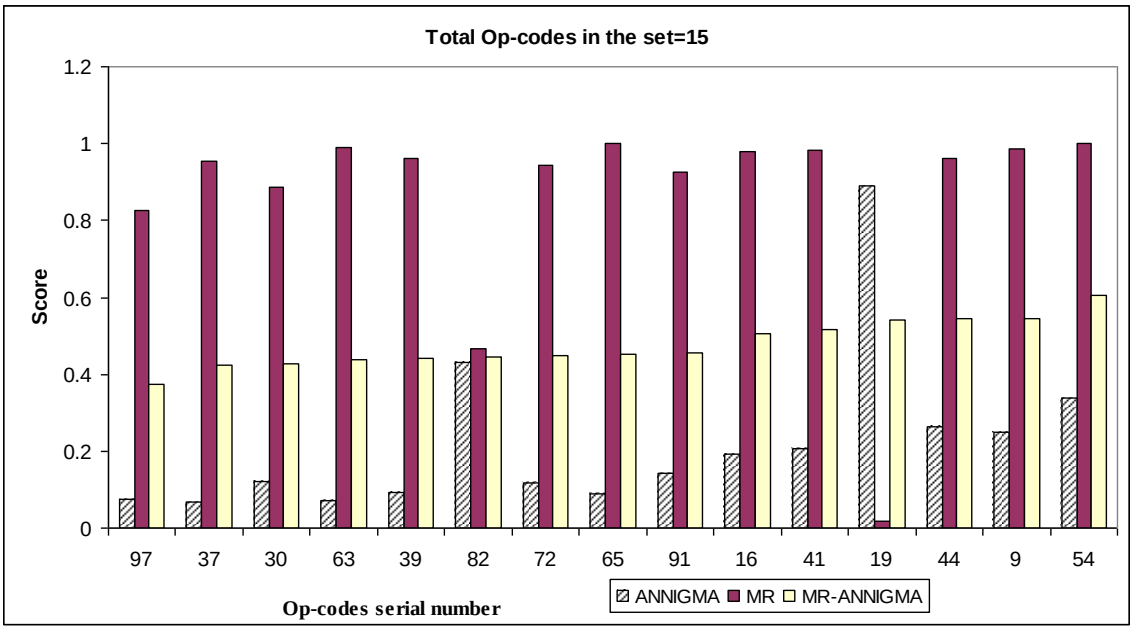


Figure 4.11 Score of OP-Codes When Total Attributes=15

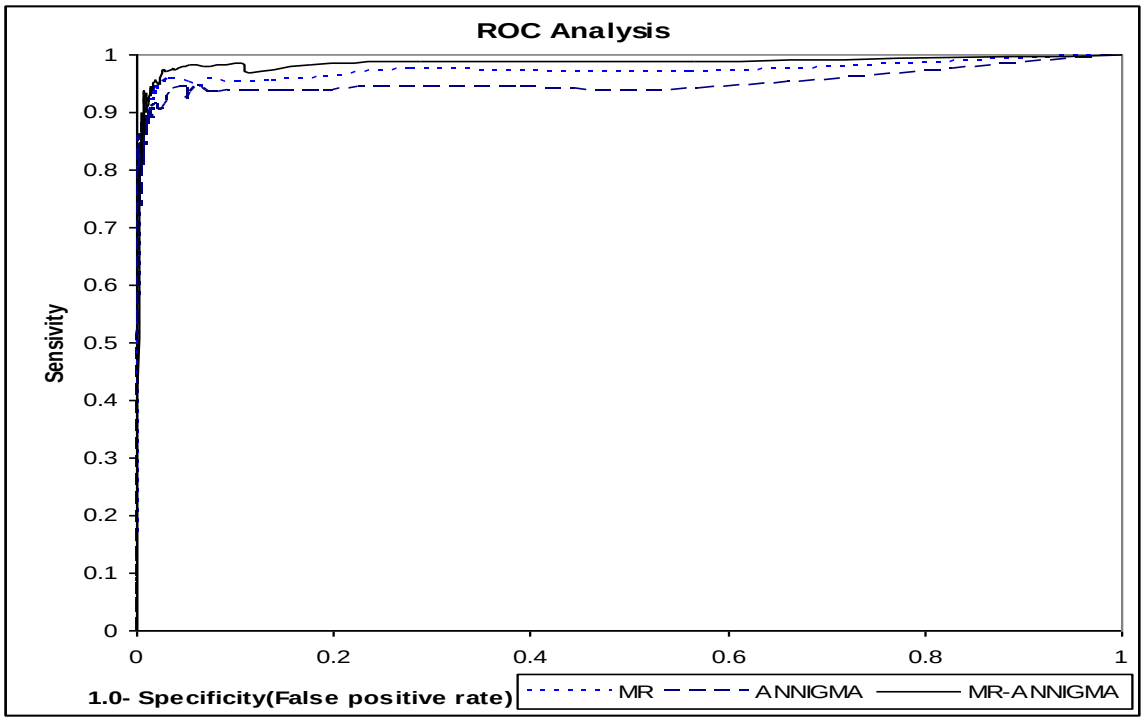


Figure 4.12 Receiver Operating Characteristics Analysis

Chapter 5 : Malware Behaviour by the Extraction of API Calls

> SHERLOCK HOLMES: *"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."*
>
> —Sir Arthur Conan Doyle, "A Scandal in Bohemia,"
>
> *The Strand Magazine* (1891)

5.1 Overview

As shown earlier, a recent technique adopted by malware authors is to use packers or software tools that instigate code obfuscation in order to evade detection by antivirus scanners. It is a common practice to undergo manual unpacking or static unpacking using existing software tools, and to use human expertise in analysing the Application Programming Interface (API) calls for malware detection. However extracting these features from the unpacked executables for reverse obfuscation is time consuming, labour intensive, and requires deep understanding of kernel and low-level programming such as assembly programming. This chapter presents an automated method of extracting API call features and analysing them in order to understand their use for malicious purpose. While some research has been conducted in arriving at file birthmarks using API call features and the like, there is a scarcity of work that analyses deeply with respect to the use of such features in malcodes. To address this gap, automatic methods to classify malware based on the behaviour the API function calls are proposed in this chapter. The

proposed approaches also provide scope for deeper understanding of code obfuscation and to reverse engineer malware automatically.

This chapter proposes a five-step methodology for developing a fully automated system to arrive at six main categories of suspicious behaviour of API call features. Also, the methodology is devised to detect obfuscated malware by investigating the structural and behavioural features of API calls. In particular, n-gram statistical analysis of API calls is applied and experimental results with a dataset of 21942 malware and 15275 benign files have shown a promising accuracy of 96.5% for the unigram model. A preliminary analysis using support vector machine (SVM) with n-values varied from 1 to 5 is also provided. Various analysis of the methods used have considered performance measurements such as accuracy, false positives and false negatives. Overall, the main objective of this chapter is to apply SVM, train the classifier and derive an optimum n-gram model for detecting both known and unknown malware efficiently.

The chapter is organized as follows: next section explains the API calls. Sections 5.3 and 5.4, describe the background of this research, which is based on a thorough foundation on API calls and PE file formats. Section 5.5 provides the motivation of this research and the current limitations of AVs. The Section 5.6 highlights the main contributions of the study. Section 5.7 provides the methodology adopted and the system architecture of the fully-automated system implemented for this research study. Then in Section 5.9, the experimental results are discussed. Finally, the limitations and future work of this ongoing research are provided in Section 5.10.

5.2 Windows API Functions

Application Program Interface (API) enables the programs to exploit the power of Windows and malware authors also make use of API calls to perform malicious actions (Sharif *et al.* 2008). Windows API function calls fall under various functional levels such as system services, user interfaces, network resources, windows shell and libraries. Since the API calls reflect the functional levels of a program, analysis of the API calls would lead to an understanding of the behaviour of the file. Malicious code is able to disguise its behaviour by using API functions provided under Win32 environment to implement their tasks. Therefore, in binary static analysis, the focus is on identifying all the documented Windows API call features to understand the malware behaviour.

Simply, the API entitles application programs to communicate with the operating system and malware authors make use of these API functions to exploit vulnerability in the system. The Windows API, or 'WinAPI' is Microsoft's core set of application programming interfaces available in the Microsoft Windows OS and the core Win API contains approximately 2500 APIs.

In the Windows operating system, user applications rely on the interface provided within a set of libraries, such as KERNEL32.DLL, NTDLL.DLL and USER32.DLL in order to access system resources including files, processes, network information and the registry. This interface is known as the Win32 API. Applications may also call functions in NTDLL.DLL known as the Native API. The Native API functions perform system calls in order to have the kernel provide the requested service. The approach described in this chapter extracts and analyses these API call features including hooking of the system

services that are responsible to manage files. The extracted calls are confined to those that affect the files. Various features related to the calls that create or modify files or even get information from the file to change some value and information about the DLLs loaded by the malware before the actual execution are considered for the analysis. Then statistical testing on the extracted features is performed to determine the malware class based on suspicious behaviours.

APIs are divided into 3 categories: kernel, user, and GDI. Figure 5.1 shows the relationship between the DLLs. Windows has a number of main sub-system DLLs such as USER32.DLL, GDI32.DLL, ADVAPI32.DLL, NTDLL.DLL and KERNEL32.DLL For the purpose of this research, the focus is on NTDLL.DLL since it serves as the fundamental user-mode interface of the windows Native APIs. NTDLL.DLL contains the Native APIs, while KERNEL32.DLL, USER32.DLL, GDI32.DLL consist of the main Win32 Core Subsystem APIs. Native API is the set of functions exported from both NTDLL.DLL and ntoskrnl.exe. The actual implementations of the Native API calls reside in ntoskrnl.exe, which means that Native API is the most direct interface into the windows kernel.

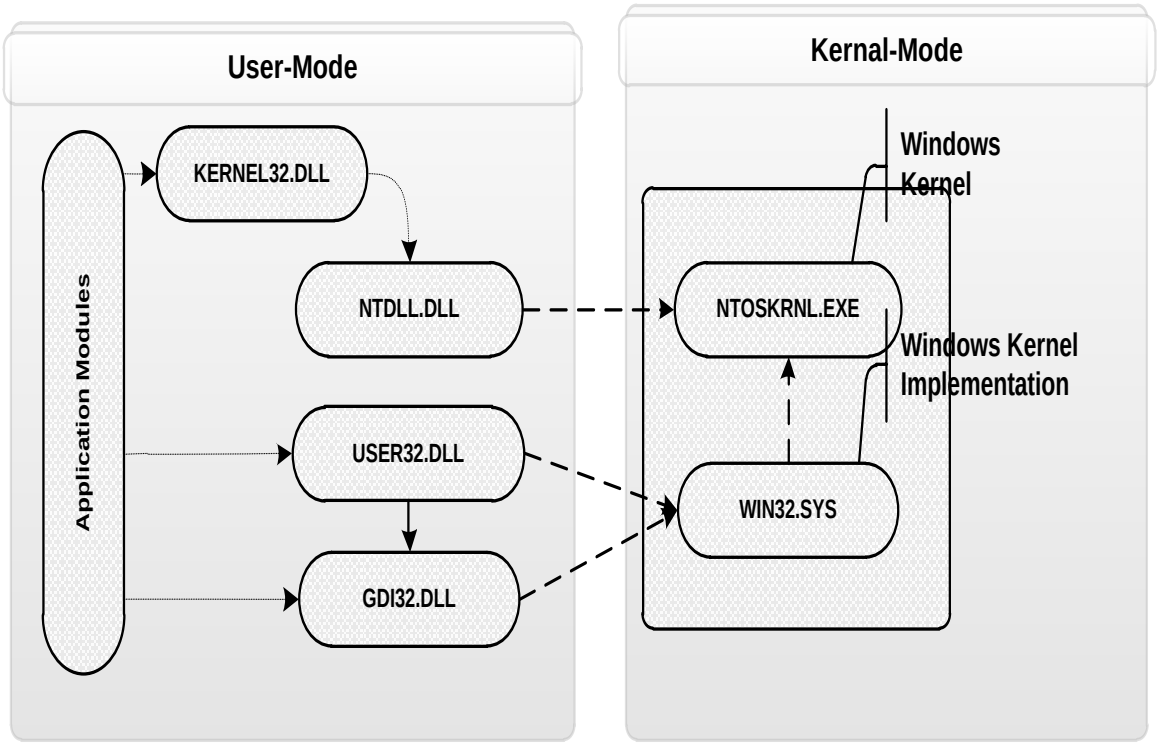


Figure 5.1 The WinAPI Interface DLLs and their Relation

The information of character strings in Portable Executable, such as file header, the number of DLL, the number of API calls and so on, could be distilled, and normal programs and virus programs are classified by naive Bayes algorithm, and the detection result reported in literature is quite good (Sun *et al.* 2010; You & Yim 2010) . However, the information in PE file header could be modified easily, and it has some difference from the true calls of programs. The existing techniques and methods do not perform sufficient statistical analyses to determine if the anomaly was 'actually' malicious. Therefore, in this research, static anomaly-based detection analysis have been used, which consists of examining the code of programs to determine properties of the dynamic execution of these programs without running them. This technique is adopted to extract

and identify the API function calls used by malware 21942 executables samples. The dataset contains recent malware samples that were collected between July 2009 and March 2010 from honeypots, honeynet project and other sources. The approach used is static analysis of the malware based on the Windows API calling sequence and this chapter describes how to extract those windows calls that reflect the behaviour of a file.

5.3 API Analysis methods

Windows OS has many undocumented APIs to give Microsoft an edge over one software vendor or another. This creates a real challenge for malware analysers where the investigation is based on the API features. However, identifying the API calls in an executable is very important since it could lead to detecting the behaviour of the Malware. In addition, it provides a sample of how the API is used and exactly what is the data it sends/receives. There are many different methods of locating APIs such as the traditional method that uses kernel mode debugger such as Numega SoftICE (Chang, H. & Atallah 2002) and WinDbg (Microsoft WinDbg 2010). Applying any of these methods in any big dataset will take a long processing time.

The Native API implemented in NTDLL forms most of the undocumented calls inside the user-mode service in Windows and hence making a dump of the export directory of NTDLL.DLL was exported to a file called a ‘dump’. The dumps from the export directories, headers and data sections of executable help to Figure out what their dependents and what the functions and services provided by NTDLL.DLL along with service numbers which are used in kernel mode. The name of documented and undocumented API and the name of the DLL that exports it are extracted and then the

binaries that use it are looked for, since MSDN library explains only default functions that Windows presents.

Offline analysis of these API calls within binary executable code is performed by using a disassembler to convert it into readable code. Offline analysis provides a better understanding of the code since the methodology adopted performs a deep analysis into the code program and their statistical properties. Therefore, in this research, static anomaly-based detection analyses have been used, which consists of an introspection of the program codes to determine various dynamic properties of these codes in an isolated environment. This technique is applied to extract and identify the API function calls

5.4 Finding Intrusions

Current anomaly-based techniques use heuristics approach of detection that is inefficient and usually results in false positives (Jacob *et al.* 2008). In this chapter, a novel approach is proposed to extract the structural and behavioural features from program codes in order to detect both known and unknown malware.

Windows API calling sequence reflects the behaviour of a particular piece of code (Ban *et al.* 2010; Wang, C., Pang, Zhao, Fu, *et al.* 2009). The API enables the programs to exploit the power of the operating system and the malware authors are using the API call as a vehicle to perform malicious actions. A novel technique of extracting the structural and behavioural features of API calls with the aid of statistical n-gram analysis has resulted in effective malware detection.

An n-gram is an n-contiguous sequence, mainly used for pattern recognition. It has been used and applied successfully in many areas of computer science applications such as Computer Speech Recognition (Xiao *et al.* 2007; Zitouni 2007), Language Identification (MacNamara *et al.* 1998), Spelling Correction (Varol & Bayrak 2011), Optical Character Recognition (OCR) (Järvelin *et al.* 2007), Authorship Analysis (Layton *et al.* 2009) etc. An n-gram method of feature extraction and analysis is quite thorough but time consuming as the size of n increases. To overcome this constraint, this chapter proposes an intelligent machine learning technique of feature recognition to train a classifier for identifying malicious code. Literature studies indicate that a predominantly used machine learning technique called, Support Vector Machines (SVMs) (Hearst *et al.* 1998) have been applied successfully to classify text, handwritings (Adankon & Cheriet ; Do & Artières 2009) and many other datasets. Since malware also exhibit the behaviour patterns in the form of a file print or feature, this work applies SVM for effectively classifying the program code as either malicious or benign.

5.5 Modern Malware Detector Issues

The results of the following recent studies have been the prime motivation for this research: 1) malware authors are able to easily fool the detection engine by applying obfuscation techniques on known malware (Sharif *et al.* 2008) , 2) identifying benign files as malware is becoming very difficult (high false positive), 3) failure to detect obfuscated malware is high (high false negative) (Symantec Enterprise Security 2010, 2011a), 4) the current detection rate is decreasing, and 5) current malware detectors are unable to detect zero day attacks (RSA 2011; Symantec Enterprise Security 1997). These

results imply that code obfuscation has become a challenge for digital forensic examiners with the limitations of signature based detection (Santos *et al.* 2009; Tang *et al.* 2010).

As a first step to addressing these issue, this chapter proposes a five-step approach of anomaly based detection that captures and analyses the structural and behavioural features of API calls from program codes or executables using n-gram and SVM to detect and classify unknown and obfuscated malware. In this chapter, a method is given on how to automate the process of effectively extracting the behaviour features of application programming interface function calls of the core of Windows operating system. The fully-automated system processes both packed and unpacked portable execution (PE) files and reverses obfuscation. This chapter discusses further the processes adopted to extract the n-gram distributions of all the API call features from the malicious and benign executables and then to apply SVM for machine learning.

The first step of this proposed method is to extract the most frequently occurring n-grams in each file and collate that list into an overall list for the entire dataset and the next step is to collect the n-gram distribution for each executable for each n-gram in the larger list. The third step is to apply principle component analysis (PCA) approach to the result to account for 95% of the variation and for an effective feature reduction process. With the extracted features, the main goal of detecting unknown malware is now possible if the proposed approach caters to all possible code obfuscation techniques that could be adopted by the malware authors.

Malware authors are continually developing such new techniques for creating malware that cannot be detected by AV engines, and their level of sophistication is

continuing to grow. Through experimental tests, it has been found that all obfuscated techniques described in Section 2.6 can be used to fool the current detection engines by obfuscating the malware signature. Hence, in order to cater to all these obfuscation techniques, this research focuses on unpacking and extracting the behaviour of the malware through API call analysis rather than the typical "pattern matching" detection process that are evaded by obfuscations of the byte sequence through packing, metamorphic and polymorphic techniques. The features of the extracted API calls are identified in the unpacked executable binary using the n-gram statistical analysis that is described in the next section.

5.6 Contributions of the Chapter

Recently, API calls have been explored for modeling program behaviour. There are studies that have used analysis of API calls for generation of a birthmark on portable execution (Choi *et al.* 2009; Park *et al.* 2008a, 2008b; Tamada *et al.* 2006). The use of statistical analysis of file binary content including statistical n-gram modeling techniques, have been tested in identifying malware in document files and does not have sufficient resolution to represent all class of file types (Stolfo *et al.* 2005; Wang, C., Pang, Zhao & Liu 2009). From other study on related work it has been found that the statistical modeling of hidden malcode that predominantly use Windows API calling sequence for evading detection is yet to be explored (Bruschi *et al.* 2006; Chang, H. & Atallah 2002; Ferrie & Szor 2001; Linn & Debray 2003; Perriot & Ferrie 2004; Venkatraman 2009). This is the motivation for this research towards a positive contribution in understanding malware behaviour through statistical analyses of API calls.

In this chapter, the static analysis tool IDAPro disassembler (IDA Pro 2010) is used to disassemble, analyze and extract the API function calls from the binary content of malware, and to statically identify the behaviour from the API calls. A novel approach is presented to automate and extract the API function calls from the malware binary content. A static anomaly based detection technique is applied and it consists of examining the malware programs without it being executed, so as to determine the behaviour of the actual execution, and to have it combine with API call feature extraction for reflecting the overall behaviour of a file.

There are four other main contributions. First, the development of a fully-automated system to unpack, de-obfuscate and reverse engineer the program codes and apply feature extraction techniques effectively. Second, the intelligent extraction of the behaviour of features of API calls that relate to i) hooking of system services, ii) creating or modifying files, iii) getting information from the file for changing information about the DLLs loaded by the malware. Third, measurement based application of n-gram statistical modeling to obtain the distribution of the executables for n-values ranging from 1 to 5. The model is measured based on factors such as accuracy, false positives and false negatives. Fourth, robust identification of malicious code as against benign code using SVM to train the classifier for machine learning.

5.7 Proposed Approach and Implementation

This section describes the methodology adopted for the automation of API extraction, the analysis and the identification of malicious behaviour. The proposed approach shown in Figure 5.2 consists of five steps for the automated detection of malware using API calls:

Step 1: Unpack the malware and disassemble the binary executable to retrieve the assembly program.

Step 2: Extract API calls and important machine-code features from the assembly program.

Step 3: Map the API calls with MSDN library and analyse the malicious behaviour.

Step 4: Extract binary n-gram features.

Step 5: Train a classifier and build a model using support vector machine.

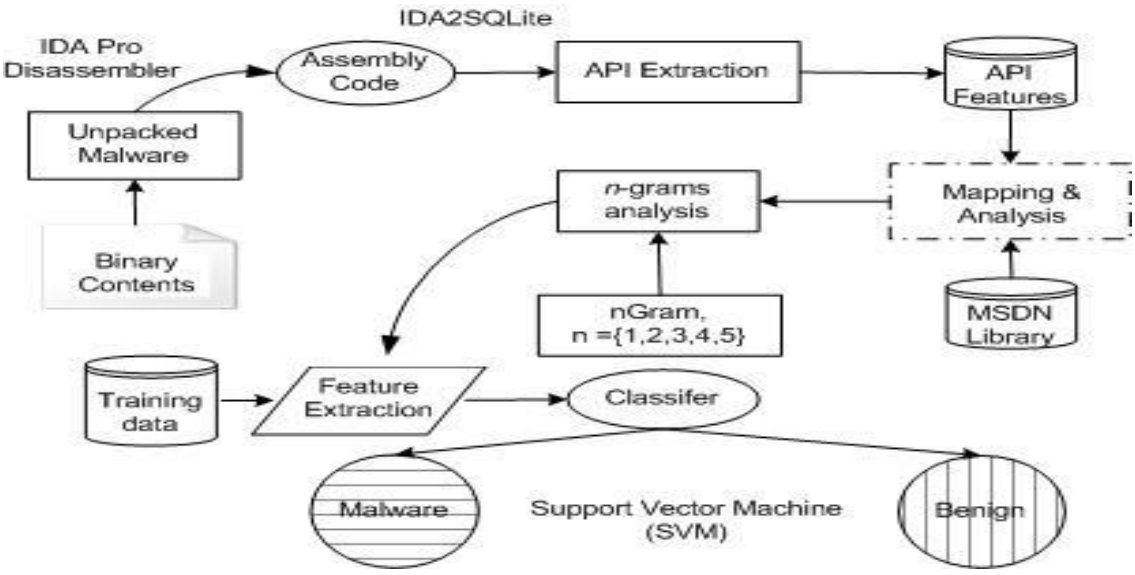


Figure 5.2 Fully-Automated Architecture to Distribute the API Function Calls

5.7.1 Step 1: Unpack and Disassemble Malware.

Researchers have been trying to build semi-automated tools for automatically unpacking malware, such as PolyUnpack (Royal *et al.* 2006), Renovo (Kang *et al.* 2007), OmniUnpack (Martignoni *et al.* 2007), Ether (Dinaburg *et al.* 2008), and Eureka (Sharif

et al. 2008). PolyUnpack is an automated unpacking technique for extracting the hidden code through process execution and uses the Windows debugging API to single-step. The second tool, Renovo, is implemented using the QEMU machine emulator and supports multiple layers of unpacking (QEMU can run an unmodified target operating system and all its applications in a virtual machine). However, OmniUnpack uses a coarse-grained execution tracking approach at the page-level protection mechanism available in hardware in order to identify when the code gets executed from a page that was newly modified. Eureka, is similar to OmniUnpack except that Eureka tracks execution at the system call level. Eureka follows a statistical bigram analysis and coarse-grained execution tracing method and provides several Windows API resolution techniques that identify API calls based on their functionality in the unpacked code. Lastly, Ether, is based on an application of hardware virtualization extension such as Intel VT, and resides outside the operating system. By studying these semi-automated tools, it is observed that none of them are completely meet the purpose of analysing the behaviour of automatically malware by extracting API call features.

In the experiment conducted on 21942 samples of malware, PEiD (PEiD 2008) was used as it is a detector adopted by most common packers, cryptors, compilers and even signature-based packer detection in PE files. The results indicate that about 77% of malware are packed and 23% are unpacked, as shown in Figure 5.3. From the result in Figure 5.3, it can be concluded that the majority of malware change their byte sequence or 'Signature' by applying packing techniques to evade detection by anti-virus scanners.

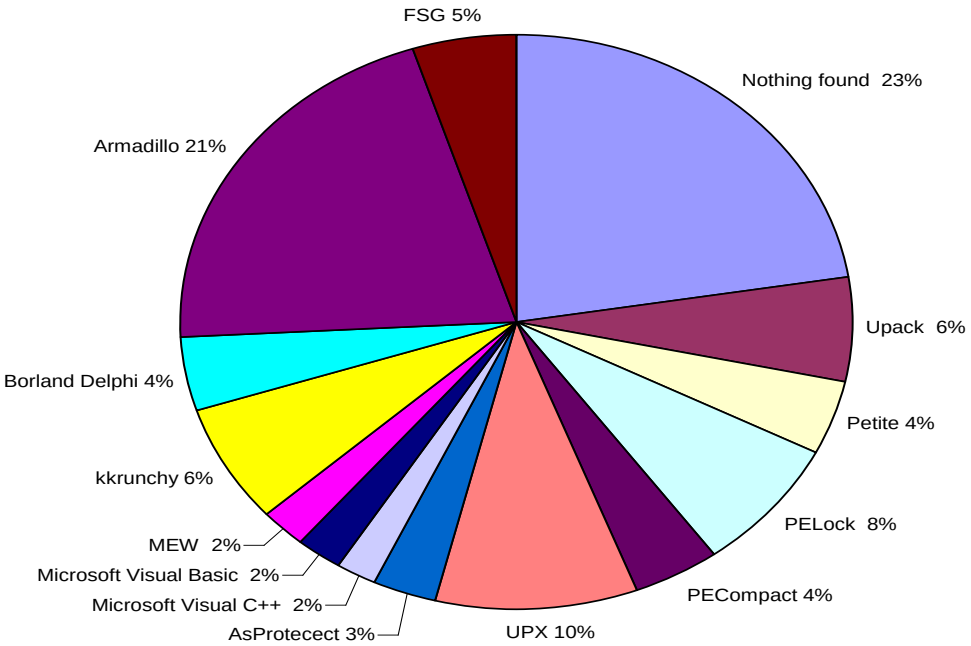


Figure 5.3 Distribution of Obfuscation Packers Used in Malware

5.7.2 Step 2: Extract API Function Calls Features.

The unpacked malware from Step 1 with the aid of SQLite and IDA pro (IDA Pro 2010) to generate a database containing the application programming interface (API) calls (.idb) automatically from the entire dataset of malware and benign programs. API call features are extracted from the assembly code of the executables so that the generated information could be used for effective analysis, using IDA pro from the unpacking.

IDA Pro provides access to its internal resources via an API that allows users to create plug-ins to be executed by IDAPro. The Python program has been created to automatically run and create the plugin to use SQLite with IDA Pro for generating the

database (.db). An interface is developed for accessing the database file (.db) so that the results from the assembly code of the malware stored in the database could be used for better binary analysis. The plugin, nameD IDASQLit is created and used with IDA pro Dissasember to generate eight tables of information, namely **`Blocks, Functions, Instructions, Names, Maps, Stacks, Segments, TargetBinaries`**. Each of these tables contains different information about the binary content. Function table contains all the recognizable API system calls and non-recognizable function names and the length (start and the end location of each function). **`Instructions table`** contains all the operation code and their addresses and block addresses. **`Maps table`** contains the function address and source of block address and the destination of the function address. **`Names table`** contains function addresses, the name of the function and the type of the function. **`Stacks table`** contains function address, the **`stack name`**, and the start and the end address. **`Segments table`** contains information that describes each segment in an executable file, segment name (`Code, Data, BSS, _idata, _tls, _rdata, _reloc, and _rsrc`) and the segment length. Finally, **`TargetBinaries`** contain the file name, path name, MD5, and start and the end of running.

5.7.3 Step 3: Map the API Function calls with MSDN Library.

Using the downloaded Windows API from Microsoft Developer Network (MSDN) (Microsoft Developer Network 2011), the proposed approach of the system compares and matches API calls outputted with the look-up table of API libraries from MSDN. The required processes are implemented in Python to compare and match the

API from MSDN and the API calls generated in the database (.db of Step 2) for the malware sample set. Another Python program was written to extract features such as the frequency of each API call, call sequence patterns and actions immediately preceding or after the call. Specific actions are considered that lead to invalid memory references or undefined registers or invalid target jumps for refining the extracted API call features.

In addition, to list all the API calls that are associated with malcode and to analyse the features, the machine op-codes such as Jump and Call operations as well as the function type (import or function) are considered. SQL code snippets are as shown below. For the analysis of malware behaviour, features such as frequency of call, call sequence pattern and actions immediately preceding or after the call are being considered. Some actions that lead to invalid memory reference or undefined register or invalid jump target are helpful in refining the extracted features for analysis. The SQL command below used to the information from the tables maps and names, to get all the API calls in the PE file where the operation code (OP) is been called or jumped of the other functions of the same

```
SELECT function_address, src_block_address, name

from Maps, Names"

Where (op='call' or op='jmp')and(type='import' or type =
'function')
```

program or the API is imported from the DLL of another PE.

All executable programs, malicious or not, have the goal to perform an action using API calls. Disguised malicious code uses a different, relatively peculiar action

called suspicious behaviour. Behaviour identification is becoming a rich area to study and as explained earlier, the malware authors target their malware on the commonly used NTFS by using API functions provided under Win32 environment to implement their functions. A statistical analysis of the Windows API calling sequence reflects the behaviour of a particular piece of code. In this research work, the API call features from the binary of a program were extracted and analysed to understand their malicious behaviour and finally to classify the program as malicious or benign. The extracted features were subjected to a statistical test to determine the malware class based on suspicious behaviours. As a result of the experimental analysis on the malware samples, the objective is to identify six main groups of commonly used API function call features that are based on the malicious behaviour patterns (Table 5.1) and these are listed below:

- Search files.
- Copy/Delete files.
- Get file information.
- Move Files.
- Read /Write files.
- Change file attributes.

Figure 5.4 shows the frequency distribution of these six main groups of API calls invoked within the experimental dataset for malicious purposes. From result as showing

in Figure 5.4, it is clear that the most prominent behaviour commonly exhibited by malware is to infect file through API calls that perform read/write files.

Figure 5.4 API Call Distribution of Malware Samples

Table 5.1 Main Malicious Behaviour Groups of API Call Features

API Function Calls	Malware Category	Behaviour
FindClose, FindFirstFile, FindFirstFileEx, FindFirstFileName, TransactedW, FindFirstFileNameW, FindFirstFileTransacted, FindFirstStream, TransactedW, FindFirstStreamW, FindNextFile, FindNextFileNameW, FindNextStreamW, SearchPath.	Search Files to Infect	Behaviour 1
CloseHandle, CopyFile, CopyFileEx, CopyFileTransacted, CreateFile, CreateFileTransacted, CreateHardLink, CreateHardLink, Transacted, CreateSymbolicLink, CreateSymbolic, LinkTransacted, DeleteFile, DeleteFileTransacted.	Copy/Delete Files	Behaviour 2
GetBinaryType, GetCompressed, FileSize, GetCompressedFile, SizeTransacted, GetFileAttributes, GetFileAttributesEx, GetFileAttributes, Transacted, GetFileBandwidth, Reservation, GetFileInformation, ByHandle, GetFileInformation, ByHandleEx, GetFileSize, GetFileSizeEx, GetFileType, GetFinalPathName, ByHandle, GetFullPathName, GetFullPathName, Transacted, GetLongPathName, GetLongPathName, Transacted, GetShortPathName, GetTempFileName, GetTempPath.	Get File Information	Behaviour 3
MoveFile, MoveFileEx, MoveFileTransacted, MoveFileWithProgress.	Move Files	Behaviour 4
OpenFile, OpenFileById, ReOpenFile, ReplaceFile, WriteFile, CreateFile, CloseHandle.	Read/Write Files	Behaviour 5
SetFileApisToANSI, SetFileApisToOEM, SetFileAttributes, SetFileAttributesTransacted, SetFileBandwidthReservation,	Change File Attributes	Behaviour 6

5.7.4 Step 4: Extract Binary n-gram features.

An n-gram model (Brown *et al.* 1992), in simple terms, uses the statistical properties of n-grams for predicting the next item in a sequence. It is a subsequence of ‘n’ items from a given sequence. For this research, the items refer to the list of API calls within an executable file. An n-gram could be of different sizes: 'unigram' referred when the size is n=1; 'bigram' where the size is n= 2; size n=3 referred to "trigram; and size n=4 or more is generally called 'n-gram'. Many disciplines have applied n-gram analysis as the model is efficient and successful in solving classification problems. In this chapter, n-grams are applied to the problem of malware detection; by extracting the list of API calls contained within both packed the unpacked malware. A classifier is trained on the differences in n-gram distributions between malicious and benign executable files.

Some studies have analysed unigram and bigrams of ASCII byte values (Stolfo *et al.* 2007) and computed the frequency and variance of each gram. They have observed that applying 'unigram' analysis to Portable Document Format (PDF) files embedded with malcode are pretty effective in malware detection when compared to the COTS AV scanners. However, such studies are limited to specific document files types and do not have sufficient resolution to include all classes of file types.

As the number of n-grams is going to be very large, feature selection measures such as ASCII, UNICODE, API and others are being adopted to yield better results. For the obfuscated malware detection problem, this work applies n-gram on API call based features. The last four proposed approaches results in an effective n-gram feature extraction from API call sequences for classifying executables as malicious or benign

with the use of step 5 the use of Support Vector Machines (SVM) as the machine learning classifier.

To extract the n-gram distributions of all of the malicious and benign executables, first the frequency of each n-gram within the entire corpus is counted. Once that has been completed, reduce this list to the top 100 most frequent n-grams. The above procedure is replicated for n values between 1 and 5 inclusive.

5.7.5 Step 5: Build a Support Vector Machine Model.

The machine learning SVMs (Cortes & Vapnik 1995) are a set of related supervised learning methods used for classification and regression. SVM constructs a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification. Basically, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class, since in general the larger the margin the lower the generalization error of the classifier. The method produces a linear classifier, so its concept description is a vector of weights ~*w*, and an interceptor a threshold *b*. However, SVMs use a kernel function to map training data into a higher dimensional space so that the problem is linearly separable. It then uses quadratic programming to set ~*w* and *b* such that the hyper plane's margin is optimal, meaning that the distance is maximal from the hyperplane to the closest examples of the positive and negative classes.

In 2006, Kolter (Kolter & Maloof 2006) described the use of machine learning and data mining to detect and classify malware. Kolter tested several classifiers including, IBk, naive Bayes, support vector machines (SVMs), decision trees, boosted naive Bayes, boosted SVMs, and boosted decision trees. Kolter found that support vector

machine performed exceptionally well and fast as compared to the other classifiers. Hence, for the obfuscated malware detection system, this research adopts SVM as a classifier for the detection of hidden malware that invariably uses API call sequence.

SVMs have performed well on traditional text classification tasks, and on executable files. The supervised learning SVM method is a reliable and popular technique for data classification. SVM is considered easier to use than many machine learning approaches such as Neural Networks (NN). Hence, in this step, SVM classification is used to construct an N-dimensional hyperplane that separates the dataset into two groups, namely, 'Malware' and 'Benign'. Initially, in this step, the data is separated into two sets: training and testing data sets. Then SVM is applied to the training data with the goal to produce a model, which is then used to predict the target of the test data. In order to achieve a higher accuracy of the predictive model for generalisation, K-fold cross-validation approach is used and applied for test data, with k=10. This value is commonly used to estimate how well the trained SVM model is going to perform in the future.

For the experiment in this research work, LIBSVM tool is used (Chang, C. & Lin 2011). Both benign files and known malicious files are used to train the SVM classifier so that the model could be used to test for new obfuscated malware that evades detection from AV scanners. The verification and validation of the proposed method was performed for malware detection based on the standard measures in Section (5.8).

Among the four basic types of kernels used by SVM to map the training vectors to the N-dimensional space, the Radial Basic Function (RBF) kernel is applied, as it can handle the nonlinear cases. Classification performance has been tested based on $\frac{1}{\sigma^2}$ and

C parameters from the equation given below, where $C > 0$ is the penalty parameter of error term.

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right) \quad (5.1)$$

5.8 Verification and Validation

The proposed method for malware detection is evaluated based on the following standard measures:

a. **True Positive** (**TP**): Number of correctly identified malicious code,

b. **False Positive** (**FP**): Number of wrongly identified benign code, when a detector detects benign file as a malware.

c. **True Negative** (**TN**): Number of correctly identified benign code.

d. **False Negative** (**FN**): Number of wrongly identified malicious code, when a detector fails to detect malware.

The efficiency of the proposed method is evaluated using the following performance measures:

Positive (*P*): The predicted attribute belong to the right class.

$$P = TP + FN$$

Negative (*N*): The predicted attribute belong to the wrong class.

$$N = FP + TN$$

Overall Accuracy: Percentage of correctly identified code, given by:

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N}$$

5.9 Experimental Results

In order to conduct an experimental investigation, the methodology of the system, described in the Section (5.7) has been used to the dataset collected from honeypots, honeynet project and other sources between July 2009 and November 2009. The dataset used is about 37217 executable files in total, with 15275 benign files and 21942 malware infected files that have been uniquely named according to their MD5 value. From the experimental study, it has been observed that the overall accuracy of the classifier decreases as n increases, as shown in Table 5.2. The trend observed here could be due to the specific dataset that was used for the experimental testing. However, any generalisation of the observed trend could only be emphatically determined with larger and wider range of datasets. The initial experimental result of 96.5% accuracy for unigrams is still very promising as a benchmark for improvements in our future research work. Further to this, there are clear trends in both the false positive and false negative values with increasing values of n. While unigrams create the better n-gram models for the values shown in Table 5.2, the high false negative rate indicates that there is still work to be done on improving this value.

Table 5.2 Experimental Results from SVM Classifier Using n-grams

n-value	Accuracy	False Positives	False Negatives
Unigram	96.50%	1.91%	1.56%
Bigrams	92.99%	6.36%	0.63%
n = 3	88.22%	11.46%	0.03%
n = 4	85.99%	14.01%	0.00%
n = 5	85.03%	14.97%	0.00%

5.10 Limitations and Future Work

The automated system makes use of existing unpacking tools, such as PolyUnpack, Renovo, OmniUnpack, Ether, and Eureka that are still under research and development. If the existing tools are unable to unpack malware that uses an unknown packing algorithm, this would pose a limitation for Step 1 of the automated system. However, the proposed approach from Step 2 onwards would still work in this case by conducting a manual unpacking in Step 1. Another limitation of the proposed approach system is that Step 3 is based on the latest updates of Microsoft with the MSDN library of API call list. It is believed that this revision is done up-to-date, as MSDN library forms the main reference for the mapping of the API calls in both malware and benign files.

Future work in this area includes techniques to increase the accuracy of the system. The FP rate increases while the FN rate decreases as n increases, indicating that it

could be possible to use a boosting technique to apply a classifier model derived from a higher n value to first determine if a file appears to be malicious, then the model uses a lower n value so as to more accurately determine if this suspected file is in fact malicious.

The system call interface is the facility that the OS offers to user-mode applications. UNIX operating system has a well-documented, clearly defined set of system calls. The MINIX operating system has a system call interface consisting of only 53 routines. Everything that the MINIX operating system is capable of doing ultimately can be resolved into one or more of these system calls. The Window operating system, which refers to its system call interface as the native API of Windows, has not provided an official document for its native API. However, today Windows is the most OS commonly used. Therefore, this research is a step towards addressing malware that try to target on windows operating system, which is the main target for malware authors with the view of affecting more computer users. Hence, this research work has limitations in its application exclusively to malware that make use of Windows API calls.

Other future work entails extracting binary n-gram features to complement the API call features and to train the classifier resulting in building a model using support vector machine (SVM). Also the model needs to be tested against larger sets of malware samples for verifying the accuracy of the modelled system.

Chapter 6 : Malware Detection based on Data Mining of API calls

---

SHERLOCK HOLMES: *"Data! Data! Data! ... I can't make bricks without clay."*

—Sir Arthur Conan Doyle, "The Adventure of the Copper Beeches*,"*

The Strand Magazine (1892)

6.1 Overview

As seen in earlier chapters, code obfuscation techniques can modify the parent code to produce offspring copies which have the same functionality with different signatures. There is number of freeware programs that are being used to generate obfuscated code. These new generations of signatures are growing and at the same time the level of sophistication of malware is also increasing. However, by analysing the offspring copies using pattern recognition, the obfuscated code could be detected. In Chapter 4 and Chapter 5 we showed that using certain pattern recognition methods to analyse op-codes and API calls leads to successfully detecting zero-day or unknown malware. This chapter investigates these patterns of obfuscated code further using data mining techniques.

In Chapter 4, signature free detection has been proposed and developed based on op-code obfuscations adopted by malware writers. In Chapter 5, an automation method of extracting the API call features has been presented and to understand their use for malicious purpose has been introduced. A preliminary analysis has been provided using

support vector machine with n-gram statistical analysis of API calls by varying n-values from 1 to 5.

In this chapter, a machine learning framework is proposed and evaluated with large datasets to further investigate the preliminary observed patterns. A variety of data mining techniques are used to detect malware from benign files. The proposed framework focuses in finding the best features and building models that can classify a given program into a malware or a benign class. The approach rests on the analysis of patterns based on the frequency of occurrence of each Windows Application Programming Interface calls found in the datasets. Also, this chapter describes in detail how various data mining techniques have been adopted to classify and detect a malware based on the frequency of Windows API calls. Also, this chapter discussed similarity based detection method by employing similarity measures of Windows Application Programming Interface (API) call features as shows in Section (6.7). Different distance measures have been implemented and similarity analysis performed by using eight commonly used distance measures in vector models, namely Cosine, Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Euclidean, and Hamming distance similarity measure for Nearest Neighbor (NN).

A supervised learning approach has been adopted that uses a dataset to train, validate and test, an array of classifiers. In order to achieve the goal of developing a detection system to identify zero day malware, robust classifiers have been selected as shown in Section (6.11) namely Naive Bayes (NB) Algorithm, k−Nearest Neighbor (kNN) Algorithm, Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels; i) SMO - Normalized Polynomial Kernel Function, ii) SMO – Polynomial Kernel Function, iii)

SMO – PUK, and iv) SMO- Radial Basis Function (RBF), Backpropagation Neural Networks Algorithm, Logistic Regression, and J48 decision tree.

This chapter also provides details of how data will be collected to conduct the experimental analysis and the data mining algorithms adopted for the study will be evaluated.

6.2 Data Mining

In recent years data mining has been the focus of many malware researchers to detect unknown malware or to classify malware from benign files. Data mining is also referred to as knowledge discovery in databases. Frawley (Frawley *et al.* 1992) define it as "The nontrivial extraction of implicit, previously unknown, and potentially useful information from data". It is also defined as "The science of extracting useful information from large data sets or databases" (Hand *et al.* 2001). In this research, data mining involves the application of a full suite of statistical and machine learning algorithms on a set of features derived from malicious and benign files.

Feature can be described as the input data to the detection systems, and can be used as patterns for classification in malware detection systems. Reverse engineering results in extracted features useful for two types of detections; i) Host-Based Intrusion Detection System (HIDS) – to check, analyse and monitor the computer system internally such as extract byte sequences, ASCII, instruction sequences, and API call sequences, and ii) Network-Based Intrusion Detection system (NIDS) - to detect malicious activity by monitoring network traffic such as denial of service (DOS) attacks, port scans.

A data mining approach to malware detection is to employ statistical classification. Each classification algorithm constructs a model, using machine learning, to represent the benign and malicious classes. In this approach, a labeled training set is required to build the class models during a process of supervised learning. The key to statistical classification is to represent the malicious and benign samples in an appropriate manner to enable the classification algorithms to work effectively. Feature extraction is an important component of effective classification, and an associated feature vector that can accurately represent the invariant characteristics in the training sets and query samples is highly desirable. Classification is the process of classifying data into two or more predetermined groups based on features, it responsible in determining if binary under inspection belongs to the group of malicious programs or to the group of benign programs.

Chapter five explained how API function calls reflect the functionality of a program and, hence, is an excellent candidate for data mining to classify malicious behaviour. Also, in Chapter 5 it was noticed that using features of API for classification can provide high accuracy rate and low false alarm rate. Therefore, in the following sections different classification techniques used in this research study and analysed based on API function call features will be discussed.

6.3 Related Study

Data mining techniques for malware detection usually starts with the first step of generating a feature set. In 2005, studies reported that a temporal consistency element was added to the system call frequency to calculate the frequency of API system call

sequences (Malan & Smith 2005). Similarity measures were calculated using edit distance and measuring similarity using the intersection of sets. The first measure was on ordered sets of native API system calls, while the second one was on unordered sets. Both similarity measures based on API gave the probabilities of two peers. The drawback is that they had considered only native API call features.

Static Analysis for Vicious Executables (SAVE) is another work based on API calls made in an attempt to detect polymorphic and metamorphic malware (Sung *et al.* 2004). SAVE defines signature as an API sequence of calls and started the reverse engineering process from decompressed 16 binaries, which are then passed through a PE file parser. Next, Windows API calls were and mapped, and passed through the similarity measure module, where similarity measures such as, Euclidian distance, Sequence alignment, Cosine measure, extended Jaccard measure, and the Pearson correlation measure were used. Binary executables under inspection were classified by identifying a high similarity to a known instance of malware in the training set. Although these similarity measures enable SAVE to detect polymorphic and metamorphic malware efficiently against 8 malware scanners, their weakness is not being able to detect unknown malware.

Another signature-free system to detect polymorphic malware and unknown malware based on the analysis of Windows API execution sequences extracted from binary executable is called Intelligent Malware Detection System (IMDS) (Ye *et al.* 2007). IMDS was developed using Objective-Oriented Association (OOA) mining based classification with a large data set gathered for the experiment (29580 binary executables, of which 12214 were benign binary executables and 17366 were malicious). For

detection, a Classification Based on Association rules (CBA) technique such as Naive Bayes, SVM and Decision Tree were used. The result was compared against anti-virus software such as Norton, Kaspersky, McAfee, and Dr.Web. In 2010 the authors of IMDS had incorporated CIDCPF into their existing IMDS system, and called it CIMDS (Yanfang *et al.* 2010). CIDCPF adapted the post processing techniques as follows: first Chi-square testing was applied and insignificant rules were pruned followed by using Database coverage based on the Chi-square measure rule ranking mechanism and Pessimistic error estimation, and finally prediction was performed by selecting the best First rule. Their results were good, but involved unbalanced test data while the training data was quite balanced. The detection rate for the training set was 89.6% and the accuracy was approximately 71.4%. The testing set had a detection rate 88.2% and accuracy of 67.6%, showing further improvements is necessry.

In 2006, (Kolter & Maloof 2006) described the use of machine learning and data mining to detect and classify malicious executables. They tested several classifiers including, IBk, naive Bayes, support vector machines (SVMs), decision trees, boosted naive Bayes, boosted SVMs, and boosted decision trees. Kolter found that SVMs performed exceptionally well compared to the other classifiers. Hence, for the obfuscated malware detection system, this research adopts SVM as a classifier for the detection of hidden malware that invariably uses API call sequence.

API based features are not only good in classification of malware, but is also good in detecting injected malicious executables. DOME (Rabek *et al.* 2003) is a host-based technique that uses static analysis based on monitoring and validating Win32 API calls for detecting malicious code in binary executables. In a study on the performance of

kernel methods in the context of robustness and generalization capabilities of malware classification (Shankarapani *et al.* 2010), results revealed that analysis based on the Win API function call provides good accuracy to classify malware.

In 2010, (Shankarapani *et al.* 2010) showed that the frequency of Windows API calls can be used to classify and detect malware with good accuracy. Authors have performed a static analysis to measure the similarity for 1593 executables, of malware and benign. Two analysis methods have been used based on the frequency of occurrence of each Windows Application Programming Interface (API). First similarity analysis, computed the mean value for 3 similarity measures (Cosine measure, extended Jaccard measure) have been used on the dataset. Second used SVM machine learning kernel RBF, to classify malware and benign. However, the result of the Receiver Operating Characteristic (ROC) curve was low and the false positive rate was too high for practical usage. In the same year of 2010, (Cesare & Xiang 2010) have performed similarity analysis using string edit distances based on control flow to Identify malware variants. However, the analysis was focused on only packed malware.

Most of previous work relating to similarity based detection exhibit some drawbacks. They performed their experiments either using small datasets or small set of API calls or PE sequences, and more importantly did not give prominence to unknown malware due to obfuscation techniques adopted in existing malware families.

API based features are not only good in classification of malware, but also in detecting injected malicious executable that result in obfuscated malware. DOME (Rabek *et al.* 2003) is a host-based technique that uses static analysis based on monitoring and

validating Win32 API calls for detecting malicious code in binary executables. In a study on the performance of kernel methods in the context of robustness and generalization capabilities of malware classification (Shankarapani *et al.* 2010), results revealed that analysis based on the Win API function call provides good accuracy to classify malware. Hence, our work focuses on investigating obfuscated malware from large datasets by employing similarity based detection methods of data mining using API call features.

6.4 Methodology

This section describes the methodology, which is an extension of the methodology provided in Chapter 5. Figure 6.1 shows the overall methodology used, which consists of three groups of processes; In the first group, the first 3 steps from Chapter 5 has been used, Step 1: Unpack the malware and disassemble the binary executable to retrieve the assembly program. Step 2: Extract API calls and important machine-code features from the assembly program. Step 3: Map the API calls with MSDN library and analyse the malicious behaviour to get the API sequence from the binaries. In the second group, after obtaining the API sequence from binaries, the signature database is updated based on these API calls. This sequence is compared to a sequence or signature (from the signature database) and is passed through the similarity measure module to generate the similarity report. Different distance measures have been implemented and similarity analysis performed by using eight commonly used distance measures in vector models, namely Cosine, Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Euclidean, and Hamming distance similarity measure for Nearest Neighbor (NN).

In third group, Mutual Information (MI) based Maximum Relevance (MR) filter ranking heuristics on the set of API function calls is used for feature selection of relevant features, which provide more information about the class variable than irrelevant features. After extracting the best features from the set of API calls, supervised learning experiments have been applied that uses a dataset to train, validate and test, an array of classifiers. Nine robust classifiers have been selected for this purpose, They are, Naive Bayes (NB) Algorithm, k−Nearest Neighbor (kNN) Algorithm, The Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels i) SMO - Normalized Polynomial Kernel Function, ii) SMO – Polynomial Kernel Function, iii) SMO – PUK, and iv) SMO- Radial Basis Function (RBF), Backpropagation Neural Networks Algorithm, Logistic Regression, and J48 decision tree.

The classification methods require training data to validate the models formulated. Therefore, K-fold cross-validation has been used for evaluating the results of a statistical analysis generating an independent dataset using 10 folds. Having k=10 folds uses 90% of full data is used for training (and 10% for testing) in each fold test. Evaluation (feature selection + classification) was done inside 10-fold cross-validation loop on all Malware and benign dataset.

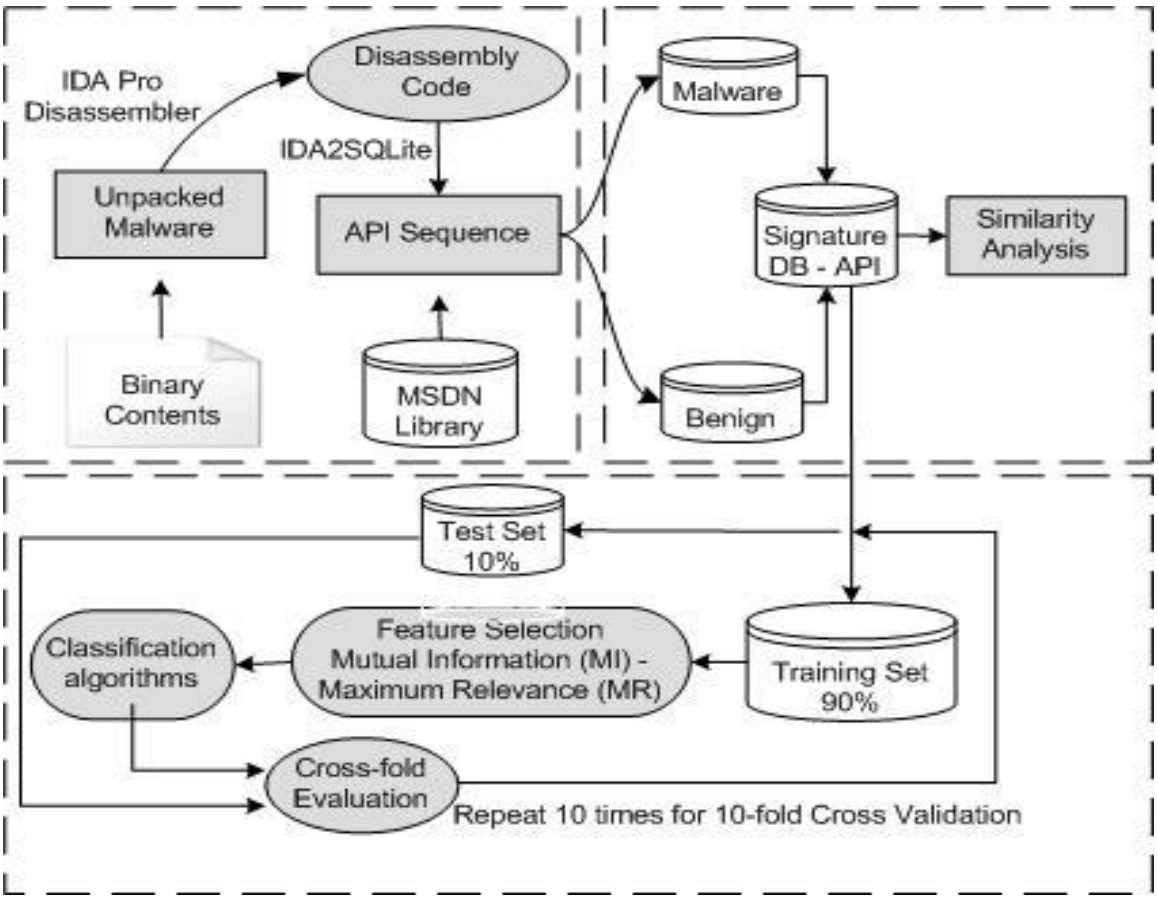


Figure 6.1 API Calls Automation, Similarity, Feature Selection and Malware Detection Methodology

6.5 Database

The dataset used in this research study consists of 66,703 executable files in total, as shown in Table (6.1). Among them, 51,223 recent Malware have been collected from honeynet project, VX heavens (VX Heavens 2011) and other sources. The remaining consists of 15,480 benign files that include : Application software such as Databases, Educational software, Mathematical software, Image editing, Spreadsheet, Word processing, Decision making software, Internet Browser, Email and many others system

software and Programming languages software and many other applications. All files have been uniquely named according to their MD5 hash value.

Table 6.1 Data set

File	Type	Qty	Max. Size (KB)	Min. Size (KB)	Avg. Size (KB)
1	Benign	15,480	109,850	0.8	32,039
2	Virus	17,509	546	1.9	142
3	Worm	10,403	13,688	1.6	860
4	Rootkit	270	570	2.8	380
5	Backdoor	6,689	1,299	2.4	685
6	Constructor	1,039	77,662	0.9	1,193
7	Exploit	1,207	22,746	0.5	375
8	Flooder	905	16,709	1	1,397
9	Trojan	13,201	17,810	0.7	1,819

6.6 Signature Generation based on API calls

Signature database has been used to statistically calculate and compute the similarity measures. Eight distance measures have been adopted to analyse and differentiate between malware variants and benign executables from various families (section 6.7). The Signature database has been generated from the database of malware and benign files (section 6.5) to produce fingerprints or benchmarks for each record based on the API calls that the executable programs had used as shown in Table 6.2.

Table 6.2 Signature Sample of API Database

ID	Called API	Class
Program 1	$\sum_{i=0}^{5} API_1 , \sum_{i=0}^{1} API_3 , \sum_{i=0}^{10} API_{16}$	Malware
Program 2	$\sum_{i=0}^{7} API_3 , \sum_{i=0}^{2} API_{11}$	Malware
Program 3	$\sum_{i=0}^{3} API_{22} , \sum_{i=0}^{4} API_{17} , \sum_{i=0}^{1} API_1 , \sum_{i=0}^{1} API_{38} , \sum_{i=0}^{n} API_6 , \sum_{i=0}^{5} API_4$	Malware
Program 5	$\sum_{i=0}^{2} API_{11} , \sum_{i=0}^{2} API_4 , \sum_{i=0}^{7} API_6 , \sum_{i=0}^{9} API_{10}$	Benign
Program 6	$\sum_{i=0}^{5} API_4$	Benign
..	…	…

The database signature has been used later in Section (6.7) to measure the distances between the programs, and also used in Section (6.11) to apply supervised machine learning algorithms.

6.7 Experiment Based on Similarity

Similarity mining is a detection method based on the analysis of similarities of the distance measures and has been adopted to detect unknown malware. Different distance measures have been implemented and similarity analysis performed by using eight

commonly used distance measures in vector models; Cosine, Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Euclidean, and Hamming distance. The maliciousness of a code is estimated using these measures. For instance, malware such as Win32.Evol (Orr 2006) has a multiple variant of the same sample malware because of the obfuscating methods adopted by the malware authors. Similarity based detection approach can be used between the variants to checked whether the variant is the child of the sample under inspection with similar features. Understanding the relationship among the distance measures can help us to choose a proper distance measure for malware detection.

Similarity based detection is well-suited for static analysis, where firstly the executable program is disassembled using reverse engineering tools. Each disassembled executable (P) represents a vector of functions x, y. (P') is the variant malware of the original executable (P). Each function is represented as an array of vector of functions. The similarity between the functions of a program (P) and (P') is computed. The value is then compared with the threshold value to determine if the executable is malicious or not.

Metamorphic and Polymorphic engines can generate and produce thousands of the malware variants. Effective engines will generate highly dissimilar copies. A 'similarity analysis' can quantify the level of similarity and the difference between two binary executables. In this section the similarity will be tested based on the extracted Win API function calls of both malware and benign files. In other words, the signature is an API sequence of known binaries that has been previously identified for both malware and benign.

For all the distance measure in the subsection below:

- u is the API sequence of the test file.
- v is the API sequence of a file in the database.
- n is the vectors dimension.
- D is the distance between vectors `u` and `v`.

6.7.1 Cosine Distance

The Cosine distance computed between two n-vectors u and v is defined as:

$$D = 1 - \frac{uv^T}{\left||u|\right|_2 \left||v|\right|_2} \tag{6.1}$$

6.7.2 Bray-Curtis Distance

The Bray-Curtis distance measured between two n-vectors `u` and `v` is defined as:

$$D = \frac{\sum |u_i - v_i|}{\sum |u_i + v_i|} \tag{6.2}$$

6.7.3 Canberra Distance

The Canberra distance between two n-vectors u and v is defined as:

$$D = \frac{\sum_i |u_i - v_i|}{\sum_i |u_i + v_i|} \tag{6.3}$$

6.7.4 Chebyshev Distance

The Chebyshev distance computed between two n-vectors u and v is defined as:

$$D = MAX_i|u_i - v_i| \quad (6.4)$$

6.7.5 Manhattan Distance

The Manhattan distance between two n-vectors u and v is defined as:

$$D = \sum_i |u_i - v_i| \quad (6.5)$$

6.7.6 Correlation Distance

The correlation distance computed between two n-vectors `u` and `v` is defined as:

$$D = 1 - frac\ (u - \bar{u}\ )\ (v - \bar{v})^T\ ||(u - \bar{u}\ )||_2||v - \bar{v}||_2^T \quad (6.6)$$

where $\bar{u}$ is the mean of a vectors elements and `n` is the common dimensionality of `u` and `v`.

6.7.7 Euclidean Distance

The Euclidean distance between two n-vectors `u` and `v` is defined as:

$$D = ||u - v||_2 \quad (6.7)$$

6.7.8 Hamming Distance

The Hamming distance between two n-vectors u and v is simply the proportion of disagreeing components in u and v, and is defined as:

$$D = \frac{C_{01} + C_{10}}{n} \quad (6.8)$$

where C_{ij} is the number of occurrence of $u[k] = i$ and $v[k] = j$ for $k < n$

6.8 Result Based Similarity Distance

The experimental investigation of the similarity analysis was carried out by implementing distance measures and analysis of the various data mining algorithms in Python Programming Language. The experiment was run in three different processors, Pentium (R) Core (TM) 2 Due CPU, 2.19 GHz, 2.98 of RAM with Windows XP professional as the operating system. The similarity analysis implemented in Python aided in the malware classification and was evaluated using very large real-life malware dataset. The datasets were obtained through public databases explained in Section (6.5). The Similarity distance system was able to automatically identify all malware variants. Similarity matrix for one set of malware in the same family are shown in Table (6.3) and Table (6.4) and the highlighted cells identifying a malware variant is defined as having a similarity equal or less 0.5. As shown the entire cell in the Table (6.3) (6.4) can be detected as a variant of the original malware Win32.Dadobra.

Similarity matrix for two malware datasets in the same families shows in Table 6.3 and Table 6.4

Table 6.3 Similarity of Trojan.Downloader.Win32.Dadobra

	.aa	.aj	.ak	.al	.am	.bf	.bh	.bw
.aa	0.000	0.100	0.207	0.100	0.138	0.138	0.138	0.143
.aj	0.100	0.000	0.226	0.000	0.226	0.226	0.226	0.233
.ak	0.207	0.226	0.000	0.226	0.207	0.207	0.207	0.276
.al	0.100	0.000	0.226	0.000	0.226	0.226	0.226	0.233
.am	0.138	0.226	0.207	0.226	0.000	0.000	0.000	0.207
.bf	0.138	0.226	0.207	0.226	0.000	0.000	0.000	0.207
.bh	0.138	0.226	0.207	0.226	0.000	0.000	0.000	0.207
.bw	0.143	0.233	0.276	0.233	0.207	0.207	0.207	0.000

Table 6.4 Similarity of Worm.Win32.Delf

	am	d	f	g	h	m	R	t	v	w	z
am	0.000	0.226	1.000	1.000	1.000	0.194	1.000	1.000	0.167	1.000	0.167
D	0.226	0.000	1.000	1.000	1.000	0.067	1.000	1.000	0.103	1.000	0.103
F	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
G	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
H	1.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000
M	0.194	0.067	1.000	1.000	1.000	0.000	1.000	1.000	0.100	1.000	0.100
R	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000
T	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000
V	0.167	0.103	1.000	1.000	1.000	0.100	1.000	1.000	0.000	1.000	0.000
W	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000
Z	0.167	0.103	1.000	1.000	1.000	0.100	1.000	1.000	0.000	1.000	0.000

Table 6.5 Similarity between Trojan.Downloader.Win32.Dadobra vs Worm.Win32.Delf

	.aa	.aj	.ak	.al	.am	.bf	.bh	.bw
am	0.100	0.188	0.290	0.188	0.167	0.167	0.167	0.200
D	0.200	0.219	0.200	0.219	0.103	0.103	0.103	0.267
F	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
G	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
H	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
M	0.167	0.188	0.226	0.188	0.100	0.100	0.100	0.233
R	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
T	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
V	0.138	0.226	0.207	0.226	0.000	0.000	0.000	0.207
W	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Z	0.138	0.226	0.207	0.226	0.000	0.000	0.000	0.207

The similarity matrix for two malware dataset from different families is given in Table 6.5. The results from Table 6.3, Table 6.4 and Table 6.5 show that there is low distance/high similarity between malware variants but not with the benign programs. Table 6.6 shows there is high distance/low similarity between the benign datasets. Table 6.7 shows the mean values for 8 different similarity measurements applied to the entire dataset, when the threshold for the similarity ratio is less than or equal to 0.5. The overall results in Table 6.7 demonstrate that the system finds high similarities between malware variants but not with the benign programs, for 8 different similarity measurements applied to the entire dataset.

Table 6.6 Similarity Matrix Benign Files

	bintext	Dic	ffind	msgr	ms hearts	msimn	ms msgs	Skin Creator	Skype	SkyTel	slide showzilla
bintext	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Dic	1.00	0.00	1.00	1.00	0.78	1.00	0.98	1.00	1.00	1.00	1.00
ffind	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
msgr	1.00	1.00	1.00	0.00	1.00	1.00	11.00	1.00	1.00	1.00	0.38
ms hearts	1.00	0.78	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00
msimn	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
msmsgs	1.00	0.98	1.00	1.00	1.00	1.00	0.00	0.81	0.98	0.98	1.00
Skin Creator	1.00	1.00	1.00	1.00	1.00	1.00	0.81	0.00	1.00	0.77	1.00
Skype	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.00	0.98	1.00
SkyTel	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.77	0.98	0.00	1.00
slide showzilla	1.00	1.00	1.00	0.38	1.00	1.00	1.00	1.00	1.00	1.00	0.00

Table 6.7 Mean similarity matrix (n × m)

Distance Method	Malware – Benign	Malware – Malware	Benign – Benign
Cosine	0.34	0.29	0.39
Bray Curtis	0.84	0.77	0.86
Canberra	0.84	0.77	0.86
Chebyshev	61.27	31.45	79.98
Manhattan	14.32	96.24	18.03
Correlation	0.35	0.243	0.403
Euclidean	78.94	44.31	106.39
Hamming	0.034	0.04	0.03

From above Tables (6.2), (6.3), (6.4), (6.5), and (6.6) it can be seen that similarity analysis is very efficient and effective to detect Malware variants from the same family or different families of Malware. Also, the experiments confirm that there is no similarity among the benign files, which is logical and true. Another important observation is that it is very hard to find similarity between the malware dataset and the benign dataset, thereby validating that the proposed system is able to clearly distinguish between malware and benign datasets. In conclusion, Malware can be classified according to similarity and further, similarly test can be applied to detect malware variants. The proposed similarity based detection and classification of malware is further improved in performance by using feature selection, especially when large datasets require more computing memory and time for processing the extracted features.

6.9 Feature Selection and Extraction

Feature selection was used to select the best subset from the input space. Its ultimate goal is to select the optimal features subset that can achieve the highest accuracy results. Many feature selection algorithms involve a combinatorial search through the whole space. Usually, heuristic methods, such as hill climbing, have to be adopted since the size of input space is exponential in the number of features. Other methods divide the feature space into several subspaces which can be searched easily.

There are basically two types of feature selection methods: filter and wrapper. Filters methods select the best features according to some prior knowledge without thinking about the bias of further induction algorithm. These methods perform independent of the classification algorithm or its error criteria.

In feature extraction, most methods are supervised. These approaches need some prior knowledge and labeled training samples. Standard filter approaches can extract knowledge of the intrinsic characteristics from real data. However, filter approaches do not use any performance criteria based on predictive accuracies. This does not guarantee that selected feature subset will be able to do better in the classification/prediction tasks.

In this step, the frequency statistics of the API calls is used to build a robust signature-free malware detection system. The filter models are based on the intrinsic characteristics of the data and do not involve the application of an induction algorithm. Diverse filter models have been advanced, including the ones that use a relevance measure, and others that deploy a distance measure as their evaluation criteria. Filter models are computationally cheap due to their evaluation criteria. However, feature subsets selected by filter may result in poor prediction accuracies, since they are independent from the induction algorithm.

For better result, there is a need to adopt feature selection approach on the set of Win API function calls on different data mining algorithms and integrating the filter's feature ranking score to find optimal feature subset for an efficient signature-free malware detection. In this approach, hybridized novel filter heuristic Mutual Information (MI) based Maximum Relevance (MR) filter ranking heuristics has been applied with different data mining algorithms.

Maximum Relevance (MR) for feature selection for relevant features provide more information about the class variable than irrelevant features. Therefore, mutual information based maximum relevance is a good heuristic to select salient features within

the data mining area. If S is a set of features F_i and class variable is c, the maximum relevance can be defined as (6.9).

$$Maximum\ Relevance\ (S,c) = \frac{1}{|S|}\sum_{f_i \in S} I(F_i;c) \qquad (6.9)$$

$I(F_i;c)$ is the mutual information between F_i and class c which is defined as (6.10).

$$I(F_i;c) = H(F_i) - H(F_i \mid c) \qquad (6.10)$$

$H(F_i)$ is the entropy of F_i with the probability density function $p(f_i)$ where F_i takes discrete values from the set $F = \{f_1, f_2 \ldots f_i\}$, then $H(F_i)$ is defined as (6.11)

$$H(F_i) = -\sum_{f_i \in F} p(f_i)\, log\ \ p(f_i) \qquad (6.11)$$

$H(F_i \mid c)$ in (6.10) is the conditional entropy between F_i and c and is defined as (6.12)

$$H(F_i \mid c) = -\sum_{f_i \in F}\sum_{c_i \in C} p(f_i, c_i)\ p(c_i \mid f_i) \qquad (6.12)$$

where class variable c takes the discrete values from the set $C = \{c_1, c_2, \ldots c_i\}$.

6.10 10-Fold Cross Validation

The classification algorithms require training data to train the formulated models, and testing data to test those models. Validation of the models is achieved by making a

partition on the database of malware and benign for carrying out the experiments. The cross-validation is a technique used for evaluating the results of a statistical analysis by generating an independent dataset for Malware and benign. The most common types of cross-validation are repeated random sub-sampling validation and K-fold cross-validation (Hand *et al.* 2001). For this research study of Malware and Benign classification, K-fold cross-validation has been selected for validation as it is commonly adopted for many classifiers (Bhattacharyya *et al.* 2011; Witten & Frank 2010).

In k-fold cross-validation the data is first partitioned into k sized segments or folds. Then, k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining k-1 folds are used for learning. The advantage of K-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing. Also, all observations are used for both training and validation, and each observation is used for validation exactly once. Extensive tests on malware and benign dataset with different learning techniques, as shown in Section (6.11), have shown that k=10 is the right number of folds to get the best estimate of error. Having 10 folds means 90% of full data is used for training (and 10% for testing) in each fold test.

6.11 Data Mining Algorithms

Classification is “the task of learning a target function that maps each feature set to one of the predefined class labels”. Data mining approach has been adopted in this research to detect malicious programs, to learn from the behaviour of existing malicious and benign database by analyzed thousands of malicious and benign programs based on Win API

features to identify and classify malware. The objective is to find out the best features and build models that can classify a given program into a malware or a benign class. First, supervised learning experiments have been applied by using a dataset to train, validate and test, an array of classifiers. In order to achieve the goal of developed a detection system to detect a zero day malware, robust data mining classifiers have been adopted, such as Naive Bayes (NB) Algorithm, k−Nearest Neighbor (kNN) Algorithm, Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels; i) SMO - Normalized Polynomial Kernel Function, ii) SMO – Polynomial Kernel Function, iii) SMO – PUK, and iv) SMO- Radial Basis Function (RBF), Backpropagation Neural Networks Algorithm, Logistic Regression, and J48 decision tree. These algorithms are described next..

6.11.1 The Naive Bayes (NB) Algorithm

The Naive Bayes algorithm (Kuncheva 2006) is one classification method based on conditional probabilities that uses a statistical approach to the problem of pattern recognition. It is reported that it is the most successful known algorithms for learning to classify text documents, and further it is fast and highly scalable for model building and scoring. The idea behind a Naive Bayes algorithm is the Bayes' Theorem and the maximum posteriori hypothesis. Bayes Theorem finds the probability of an event occurring given the probability of another event that has occurred already. For instant, for a feature vector x with n attributes values $x = x_1, x_2, \dots x_n$ and a class variable C_j, $C = c_1, c_2, \dots c_j$.

Bayesian classifiers can predict class membership C_j with probabilities $P(x|C_j)$ for the feature vector x whose distribution depends on the class C_i. The class C_j for which $P(C_j|x)$ is called the maximum posteriori probability that feature vector x belongs, can be computed from $P(x|C_j)$ by Bayes' rule:

$$P(C_j|x) = \frac{P(x|C_j)\,P(C_j)}{P(x)} \tag{6.13}$$

It applies "naïve" conditional independence assumptions which states that all n features $x = x_1, x_2, \ldots x_n$ of the feature vector x are all conditionally independent of one another, given $P(C_j)$ Naïve Bayes assumption as follows:

$$P(x|C_j) = P(x_1, x_2, \ldots x_n|C_j) = \prod_{i=1}^{n} P(x_i|C_j) \tag{6.14}$$

$$P(C_j|x) = \frac{P(C_j)\prod_{i=1}^{n} P(x_i|C_j)}{P(x)} \tag{6.15}$$

The most probable hypothesis given the training data *'Maximum a posteriori'* hypothesis results in the following:

$$C_{max} = arg\,max_{C_m}\ P(C_j) \prod_{i=1}^{n} P(x_i|C_j) \tag{6.16}$$

In data mining, Naive Bayes algorithm is easy to implement and is an efficient and effective inductive learning algorithm for machine learning.

Among data mining methods, Naive Bayes algorithm is easy to implement and is an efficient and effective inductive learning algorithm for machine learning. Figure 6.2 and Table 6.8 provide the overall accuracy rate for malware detection achieved through our experiments using Naive Bayes with k cross validations, k= {2,3,4,5,6,7,8,9,10}.

Figure 6.2 Accuracy of NB with k Cross Validations (k=2 to 10)

Table 6.8 Performance of Naive Bayes Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.922	0.082	0.926	0.922	0.924	0.938	92.02	0.23
3	0.912	0.089	0.912	0.912	0.912	0.933	91.19	0.21
4	0.923	0.077	0.923	0.923	0.923	0.938	92.3	0.29
5	0.909	0.091	0.909	0.909	0.909	0.933	90.91	0.25
6	0.917	0.084	0.917	0.917	0.916	0.94	91.65	0.22
7	0.912	0.089	0.912	0.912	0.912	0.932	91.19	0.23
8	0.915	0.086	0.915	0.915	0.915	0.938	91.47	0.23
9	0.913	0.088	0.913	0.913	0.913	0.939	91.28	0.23
10	0.91	0.09	0.91	0.91	0.91	0.938	91	0.2

6.11.2 The k−Nearest Neighbor (kNN) Algorithm

kNN is simple supervised machine learning algorithm that is used for classifying objects based on closest training instances in the feature space. It has been employed in many applications in data mining, statistical pattern recognition and many others. It has been used in many applications in data mining, statistical pattern recognition and many others. The object is classified based on a majority vote of its k nearest neighbors /low distance to the object. As showed in (section 6.7) there are some measuring techniques that could be used to measure distance between the training object and the test object such as Bray-Curtis, Euclidean, correlation, Canberra, Manhattean, Chebyshev, Dice, Cosine, and Hamming distances.

In our experiments, the K-nearest neighbors are compute as follows with K:

1. Store all training samples x_i^j in memory.

2. Determine the parameter K = number of nearest neighbors beforehand. (A good k can be selected using cross-validation for example).

3. Measure the distance between the query-instance (x) and all the training samples x_i^j. (any distance algorithm can be used to) such as:

$$dis(x, x_i^j) \sqrt{\sum_{i=1}^{d} x(i) - x_i^j\,(i)^2} \tag{6.17}$$

4. Find the K-minimum distance between the query-instance (x) and each K $j_{min}^1, j_{min}^2, \dots, j_{min}^k$.

5. Get all categories of training data for the sorted value under K.

6. Find the weighted distance of the query-instance (x) from each of the k nearest points as follows:

$$w = 1 - \frac{dis(x, x_i^j)}{\sum_{i=0}^{k} dis(x, x_i^j)} \tag{6.18}$$

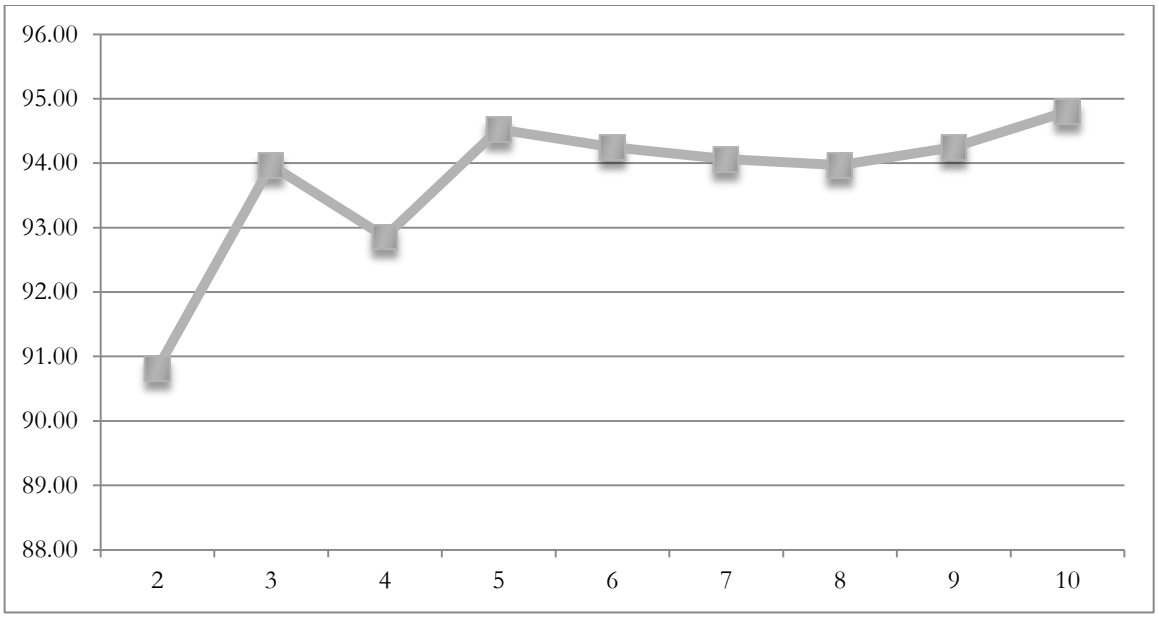


Figure 6.3 Accuracy of kNN with k Cross Validations (k=2 to 10)

Figure 6.3 provides the overall accuracy rate for malware detection achieved through our experiments using K-Nearest Neighbors with k cross validations, k= {2,3,4,5,6,7,8,9,10}. Table 6.9 shows the performance of kNN Fold Cross Validation (k=2 to 10)

Table 6.9 Performance of kNN Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.908	0.092	0.908	0.908	0.908	0.933	90.82	0.21
3	0.94	0.059	0.94	0.94	0.94	0.958	0.94	0.23
4	0.929	0.071	0.929	0.929	0.929	0.949	0.93	0.21
5	0.945	0.054	0.946	0.945	0.945	0.963	0.95	0.2
6	0.942	0.056	0.943	0.942	0.943	0.96	0.94	0.22
7	0.941	0.058	0.941	0.941	0.941	0.958	94.06	0.2
8	0.94	0.059	0.94	0.94	0.94	0.96	93.97	0.21
9	0.942	0.056	0.943	0.942	0.943	0.958	94.25	0.23
10	0.948	0.051	0.948	0.948	0.948	0.966	94.81	0.2

6.11.3 The Sequential Minimal Optimization (SMO) Algorithm

Sequential Minimal Optimization (SMO) is a simple algorithm that can quickly solve the SVM QP problem, without any extra matrix storage and without using numerical QP optimization. The advantage of SMO is its ability to solve the Lagrange multipliers analytically. SMO is a supervised learning algorithm used for classification and regression, and it is a fast implementation of Support Vector Machines (SVM). The basic advantage is that it attempts to maximise the margin, for example the distance between the classifier and the nearest training datum. SMO constructs a hyperplane or set of hyperplanes in a *n*-dimensional space, which can be used for classification. Basically, a separation can be good when the hyperplane has the largest distance to the nearest training data points of any class, since in general the larger the margin the lower the generalization error of the classifier. SMO has been selected to classify malicious and

beingn executables because it is competitive with other SVM training methods such as Projected Conjugate Gradient "chunking", and in addition it is easier to implement in WEKA (Witten & Frank 2010).

As shown in Figure 6.4, we have employed 4 different kernels; Radial Basis Function Kernel (RBF), Polynomial kernel, Normalized Polynomial kernel, and the Pearson VII function-based universal kernel (Puk), and the overall accuracy rate for malware detection achieved through Normalized Polynomial kernel is the highest for all the k cross validations, k={2,3,4,5,6,7,8,9,10}. Tables 6.10, 6.11, 6.12 and 6.13 shows the performance of the four different kernel of SMO Fold Cross Validation (k=2 to 10).

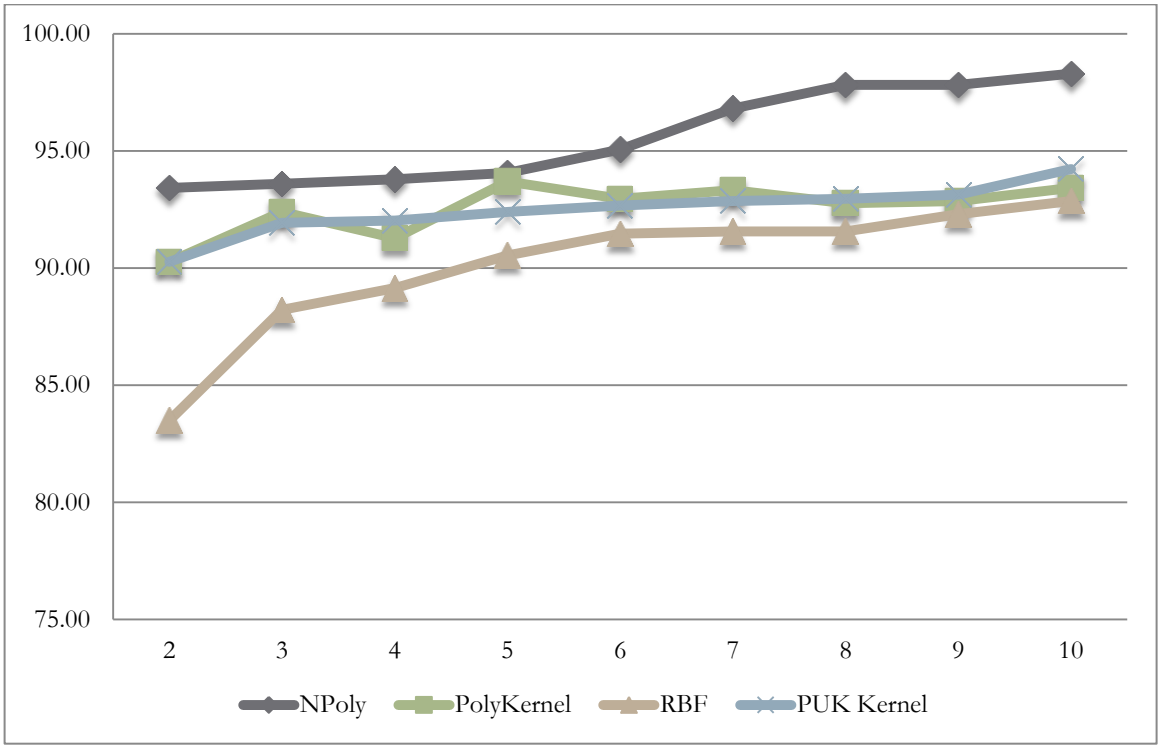


Figure 6.4 Accuracy of SMO with k Cross Validations (k=2 to 10)

Table 6.10 Performance of SMO Normalized Polynomial Kernel Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.934	0.063	0.937	0.934	0.934	0.936	93.41	4.46
3	0.936	0.061	0.939	0.936	0.936	0.938	93.6	4.34
4	0.938	0.059	0.94	0.938	0.938	0.94	93.78	4.4
5	0.941	0.056	0.943	0.941	0.941	0.942	94.06	4.47
6	0.953	0.042	0.953	0.951	0.951	0.952	95.06	4.3
7	0.968	0.043	0.968	0.968	0.968	0.966	96.81	4.37
8	0.977	0.051	0.978	0.977	0.977	0.966	97.82	4.33
9	0.971	0.044	0.968	0.971	0.971	0.996	97.82	4.29
10	0.986	0.025	0.976	0.986	0.984	0.982	98.3	4.01

Table 6.11 Performance of SMO Polynomial Kernel Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.903	0.104	0.908	0.903	0.902	0.899	90.26	1.98
3	0.924	0.081	0.927	0.924	0.924	0.921	92.39	1.83
4	0.913	0.092	0.916	0.913	0.912	0.91	91.28	1.81
5	0.937	0.067	0.939	0.937	0.937	0.935	93.69	1.8
6	0.929	0.075	0.932	0.929	0.929	0.927	92.95	1.89
7	0.933	0.07	0.935	0.933	0.933	0.931	93.32	2.07
8	0.928	0.077	0.93	0.928	0.927	0.925	92.76	1.84
9	0.929	0.076	0.931	0.929	0.928	0.926	92.86	1.84
10	0.934	0.069	0.936	0.934	0.934	0.932	93.41	1.8

Table 6.12 S Performance of SMO PUK Kernel Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.903	0.093	0.907	0.903	0.903	0.905	90.26	4.3
3	0.919	0.077	0.923	0.919	0.919	0.921	91.93	4.1
4	0.92	0.077	0.922	0.92	0.92	0.922	92.02	4
5	0.924	0.072	0.927	0.924	0.924	0.926	92.39	3.95
8	0.927	0.07	0.929	0.927	0.927	0.928	92.67	3.4
6	0.929	0.068	0.931	0.929	0.929	0.93	92.86	3.71
7	0.929	0.067	0.932	0.929	0.93	0.931	92.95	3.52
9	0.931	0.065	0.934	0.931	0.931	0.933	93.14	3.17
10	0.94	0.064	0.94	0.932	0.932	0.939	94.23	3.1

Table 6.13 Performance of SMO RBF Kernel Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.835	0.154	0.855	0.835	0.834	0.84	83.4879	3.05
3	0.882	0.113	0.886	0.882	0.882	0.884	88.22	2.73
4	0.891	0.106	0.894	0.891	0.892	0.893	89.15	2.71
6	0.905	0.094	0.906	0.905	0.905	0.906	90.54	2.98
9	0.915	0.086	0.915	0.915	0.915	0.914	91.47	2.8
5	0.916	0.084	0.916	0.916	0.916	0.916	91.5584	2.8
7	0.916	0.085	0.916	0.916	0.916	0.915	91.56	2.69
8	0.923	0.078	0.923	0.923	0.923	0.922	92.30	2.67
10	0.929	0.073	0.929	0.929	0.929	0.928	92.86	2.6

6.11.4 Artificial Neural Networks (ANN) Algorithm

Artificial Neural Networks (ANN) are biologically inspired form of distributed computing usually comprising of a set of nodes (including input, hidden and output) and weighted connections between them (Chen *et al.* 2005). Guo and Li define ANNs as a topology/architecture formed by organizing nodes into layers and linking the layers of neurons (Guo & Li 2008). The nodes are interconnected by weighted connections, and the weights are adjusted when data is presented to the network during a training process (Dayhoff & DeLeo 2001)

A number of variations of neural networks are in use today in different applications, including in fraud detection. The use of ANNs in fraud detection spans almost all major forms of fraud including telecommunications fraud, financial fraud and computer intrusion fraud among others (Kou *et al.* 2004). In fraud detection and anomaly detection, ANNs are fundamentally used as classification tools (Chandola *et al.* 2009). Usually, a basic anomaly detection approach using neural networks involves two steps: training and testing. First, the network is trained on some part of the data to learn the different classes. Then, the remaining portion of the data is used to run the network to test accuracy and other performance indicators.

ANNs provide a non-linear mapping from the input space to the output space so can learn from the given cases and generalize the internal patterns of a given data set (Guo & Li 2008). Thus, ANNs adapt the connection weights between neurons and approximate a mapping function that models the provided training data. Neural networks have the ability to learn distinct classes without knowledge of the data distribution

(Chandola *et al.* 2009). However, most classifications rely on accurately labelled data which is often not readily available, especially for online banking and credit card fraud detection (Chandola *et al.* 2009). In Credit Card fraud detection, the FALCON system, which the developers claim to be used by 65% of the credit systems worldwide, uses a Neural network (FICO 2010). Furthermore, VISA, Eurocard and Bank Of America (among others) use Neural technology in their Credit Card systems (Aleskerov *et al.* 1997). The SAS fraud management system employs an ensemble of neural networks called Self Organizing Neural Network Arboretum (SONNA). Lastly, ACI's Proactive Risk Manager (PRM) also features a neural network in its architecture (IBM 2008).

The downside to neural networks' distribution free generalisation is that they are prone to local minima and over-fitting (Bhattacharyya *et al.* 2011). When the ANN is learning, a stopping condition may be declared as the anticipated net training error after a training session. This value is often a global minimum relative to the network's training errors. Sometimes, the ANN stops learning and gets stuck at a local minimum instead of the desired global minimum. This situation is most commonly referred to as the local minimum problem. Another problem with ANNs is hidden neuron saturation, where the hidden layer inputs are too high or too low such that the hidden layer output is almost close to the bounds of the activation function at that layer (Wang, X. G. *et al.* 2004). The other drawback with ANNs is their lack of adaptation to new data trends. At any point, ANNs will model only the data they have been trained on. This means that when a statistically different data pattern is introduced, the ANN will need to be re-trained or it may not correctly classify the new pattern. Consequently, this dictates that ANNs be retrained on a regular basis to keep up with emerging data trends. In online banking,

ANNs are retrained after a defined period or after a certain number of examples have been collected.

Recently, classification method using a NN was used for Malware detection. Generally, the classification procedure using the NN consisted of three steps, data preprocessing, data training, and testing. The data preprocessing was for the feature selection. In the data training, the selected features from the data preprocessing step was fed into the NN, and Malware and Benign classifier was generated through the NN. For the testing, the classifier was used to verify the efficiency of NN. In the experiment, an error BP (Back Propagation) algorithm was used. The best-known example of a neural network training algorithm, namely back propagation was employed. Back propagation algorithm neural network was used because of the large amount of input/output data and the overwhelming amount of complexity due to the fuzzy outputs.

Figure 6.5 provides the overall accuracy rate for malware detection achieved through our experiments using Artificial Neural Networks with k cross validations, k= {2,3,4,5,6,7,8,9,10}. Table 6.14 shows the performance of ANN Fold Cross Validation (k=2 to 10)

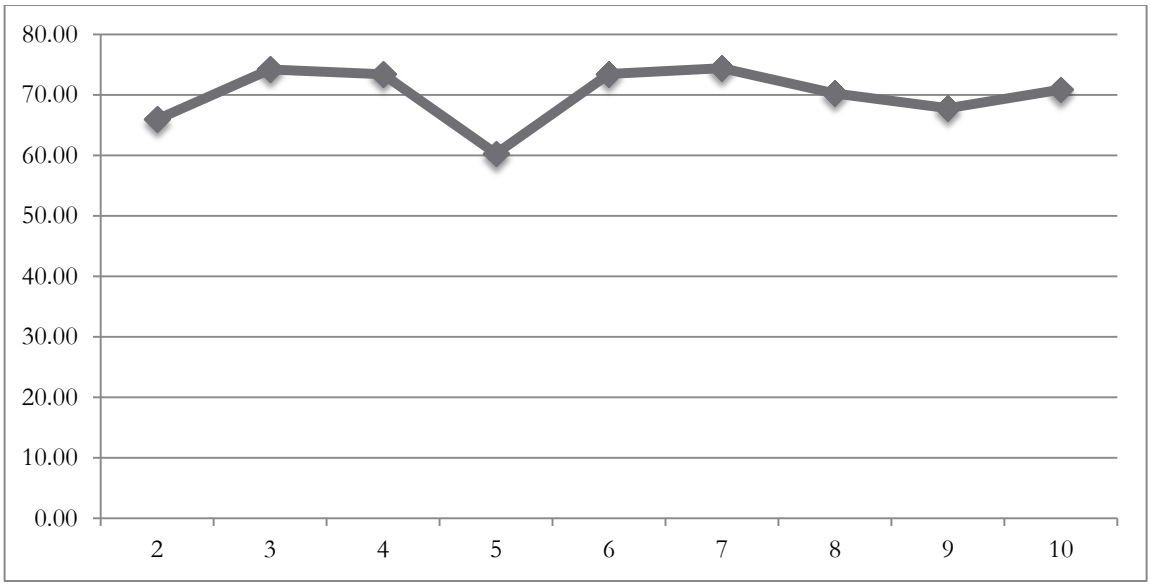


Figure 6.5 Accuracy of ANN with k Cross Validations (k=2 to 10)

Table 6.14 Performance of Artificial Neural Networks Fold Cross Validation (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.857	0.248	0.882	0.857	0.847	0.742	65.96	16.26
3	0.888	0.175	0.904	0.888	0.883	0.8	74.21	15.6
4	0.827	0.244	0.863	0.827	0.815	0.77	73.38	15.52
5	0.81	0.198	0.856	0.81	0.803	0.725	60.30	15.46
6	0.828	0.235	0.865	0.828	0.817	0.773	73.47	15.43
7	0.836	0.223	0.871	0.836	0.827	0.782	74.40	15.44
8	0.832	0.213	0.869	0.832	0.823	0.768	70.22	15.43
9	0.798	0.263	0.848	0.798	0.783	0.738	67.81	16.17
10	0.864	0.183	0.881	0.864	0.859	0.783	70.87	15.43

6.11.5 Logistic Regression

Logistic regression is a member of the family of methods called *generalized linear models* ("GLM"). The linear function of the predictor variables is calculated, and the result of this calculation is run through the link function. In logistic regression, the linear result is run through a *logistic function*, which runs from 0.0 (at negative infinity) and rises monotonically to 1.0 (at positive infinity). Along the way, it is 0.5 when the input value is exactly zero. Among other desirable properties, it is interesting to note that this logistic function only returns values between 0.0 and 1.0. Other GLMs operate similarly, but employ different link functions- some of which are also bound by 0.0 - 1.0, and some of which are not.

For building linear logistic regression models, Lotus (Chan & Loh 2004) is a logistic regression tree learner for two class problems (Malware and Benign). The algorithm constructs logistic regression trees in a top-down way, emphasizes the importance of unbiased split variable selection through the use of a modified chi-square test, and uses only numeric attributes for constructing logistic models.

In univariate regression equation, say y = a + b x + e, there is only one explanatory (or independent variable) x, while the other components in this relation are the response (or dependent variable) y, the coefficients a and b and the error term e. The response usually is a continuous random variable, for example Gaussian responses, which is the most popular case in regression, whilst in some other cases the response can take discreet values. Logistic regression is one example of these cases. The general graph of this model is represented by the following plot.

The response (or dependent variable) in logistic regression is usually dichotomous (i.e. measured at two levels), that is, it can take the value 1 with a probability of success p, or the value 0 with probability of failure 1-p. The logistic model in multivariate shape can take the following equation:

$$logit\,[\,p(x)\,] = log\left[\frac{p(x)}{1-p(x)}\right]$$

$$= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n \qquad (6.19)$$

where the constants $\beta_0, \beta_1, \ldots, \beta_n$ are called coefficients and $X_1, X_2, \ldots, X_n$ are the "multivariate" explanatory variables. The detection (MLW) data are coded and considered as response variable y_i while all other features are considered as independent or explanatory variables.

$$\mathrm{MLW_i} = \begin{cases} 1 & \text{if the i}^{\text{th}}\text{ is Malware} \\ 0 & \text{if the i}^{\text{th}}\text{ is Benign} \end{cases} \qquad (6.20)$$

The logistic regression model is

$$\log\left[\frac{p(\mathrm{MLW_i} \mid x_1, x_2, \ldots, x_n)}{1-p(\mathrm{MLW_i} \mid x_1, x_2, \ldots, x_n)}\right]$$

$$= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n \qquad (6.21)$$

where $x_1, x_2, .. x_n$ is the feature of API function calls, and the risk of malware is defined in (6.22)

$$P(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n)}} \qquad (6.22)$$

The probability of a file to be a Malware is defined as follows:

$$P(X) = \frac{e^{(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}{1+e^{(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}} \quad (6.23)$$

The goal of logistic regression is to correctly predict the category of outcome for individual cases using the most parsimonious model. To achieve this aim, a model is created that includes all independent variables (or explanatory variables) that are significant (useful) in predicting the response variable. Several different options are available during model creation. Variables can be entered into the model in the order specified by the researcher or logistic regression can test the fit of the model after each coefficient is either added or deleted, called stepwise regression. In R programming software, the command ***Step()*** can do this depending on Akaike's Information Criterion (AIC).

Figure 6.6 provides the overall accuracy rate for malware detection achieved through our experiments using Logistic Regression with k cross validations, k= {2,3,4,5,6,7,8,9,10}. Table 6.15 shows the performance of Logistic Regression Fold Cross Validation (k=2 to 10).

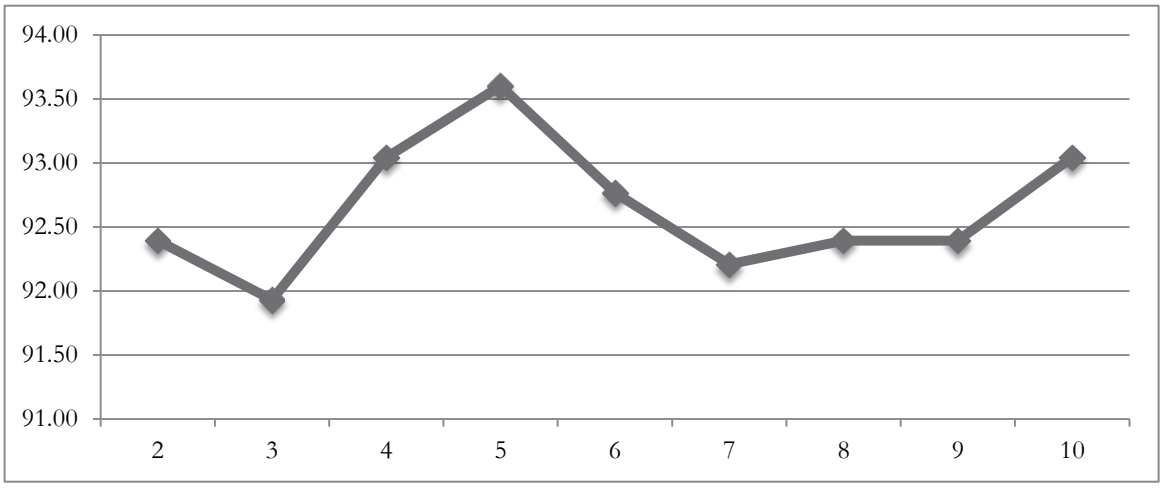


Figure 6.6 Accuracy of Logistic Regression with k Cross Validations (k=2 to 10)

Table 6.15 Performance of Logistic Regression with k Cross Validations (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.924	0.076	0.924	0.924	0.924	0.958	92.39	70.53
3	0.919	0.081	0.919	0.919	0.919	0.963	91.93	70.27
4	0.93	0.07	0.93	0.93	0.93	0.968	93.04	71.16
5	0.936	0.064	0.936	0.936	0.936	0.969	93.60	51.79
6	0.928	0.073	0.928	0.928	0.928	0.969	92.76	50.88
7	0.922	0.078	0.922	0.922	0.922	0.97	92.21	51.08
8	0.924	0.076	0.924	0.924	0.924	0.965	92.39	51.21
9	0.924	0.076	0.924	0.924	0.924	0.968	92.39	50.85
10	0.93	0.07	0.93	0.93	0.93	0.97	93.04	52.69

6.11.6 J48

J48 classifier is a C4.5 decision tree used for classification purposes. In order to classify a new item, the classifier first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly. This feature that is able to tell the most about the data instances for classifying them the best is said to have the highest information gain.

Among the possible values of this feature, if there is any value for which there is no ambiguity, that is, when the data instances falling within its category have the same value for the target variable, then that branch is terminated and the target value arrived is

assigned to it. Figure 6.7 provides the overall accuracy rate for malware detection achieved through our experiments using J48 with k cross validations, k= {2,3,4,5,6,7,8,9,10} and Table 6.16 provides the overall accuracy rate for malware detection achieved through our experiments using Naive Bayes with k cross validations, k= {2,3,4,5,6,7,8,9,10}.

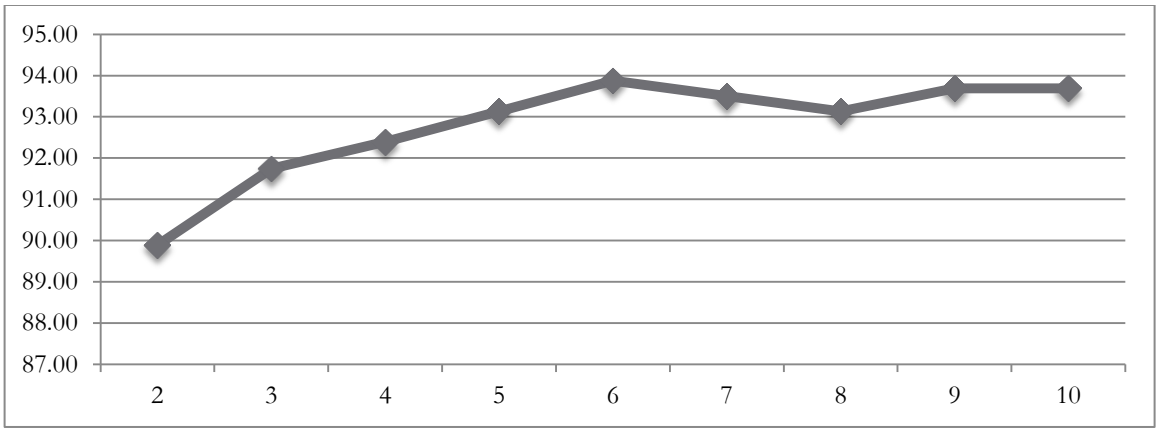


Figure 6.7 Accuracy of J48 with k Cross Validations (k=2 to 10)

Table 6.16 Performance of J48 with k Cross Validations (k=2 to 10)

Fold	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Accuracy	Time Sec.
2	0.899	0.1	0.899	0.899	0.899	0.891	89.89	2.02
3	0.917	0.082	0.918	0.917	0.917	0.919	91.74	1.9
4	0.924	0.074	0.925	0.924	0.924	0.923	92.39	1.79
5	0.931	0.067	0.932	0.931	0.931	0.93	93.14	1.86
6	0.939	0.06	0.939	0.939	0.939	0.935	93.88	1.65
7	0.935	0.064	0.936	0.935	0.935	0.942	93.51	1.67
8	0.931	0.068	0.932	0.931	0.931	0.929	93.14	1.64
9	0.937	0.062	0.937	0.937	0.937	0.946	93.69	1.68
10	0.93	0.068	0.931	0.93	0.93	0.931	93.69	1.67

6.12 Evaluation and Validation Metrics

Several classification techniques have been used in this research study and for comparing the performance of each algorithm, it is important to assess how good a classification algorithm is able to correctly predict variables. The proposed method for malware detection would be evaluated based on the following standard measures:

True Positive (*TP*): Number of correctly identified malicious code,

False Positive (*FP*): Number of wrongly identified benign code, when a detector identifies a benign file as a malware.

True Negative (*TN*): Number of correctly identified benign code.

False Negative (*FN*): Number of wrongly identified malicious code, when a detector fails to identify the malware because the virus is new and no signature is available yet.

The efficiency of the proposed method would be evaluated using performance measures such as detection rate, false alarm rate and overall accuracy that are defined as follows:

Positive (*P*): The predicted attribute belong to the right class.

$$P = TP + FN \tag{6.24}$$

Negative (*N*): The predicted attribute belong to the right class.

$$N = FP + TN \tag{6.25}$$

True detection Rate (*TP rate*): Percentage of correctly identified malicious code.

$$TP\ Rate = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{6.26}$$

False alarm Rate (*FP rate*): Percentage of wrongly identified benign code, given by:

$$FP\ Rate = \frac{FP}{FP + TN} = \frac{TP}{N} \tag{6.27}$$

Overall Accuracy: Percentage of correctly identified code, given by:

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N} \tag{6.28}$$

Two other fundamental measures used for classification effectiveness are precision and recall.

The precision: the probability of records classified as positive which are classified correctly.

$$Precision = \frac{TP}{TP + FP} \tag{6.29}$$

The recall: Recall in this context is also referred to as the True Positive Rate. It is the probability of positive records that have been correctly identified.

$$Recall = \frac{TP}{TP + FN} \tag{6.30}$$

F-Measure: It is a measure of a test's accuracy by combining recall and precision scores into a single measure of performance, usually it is between 0.0 and 1.0, closer to 1 being a good score and closer to 0.0 being a poor score.

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6.31}$$

The Receiver Operating Characteristic (ROC) curve: In a ROC curve the true positive rate is plotted in function of the false positive rate (100-Specificity) for different cut-off points. Each point on the ROC curve represents a sensitivity/specificity pair corresponding to a particular decision threshold. A test with perfect discrimination (no overlap in the two distributions) has a ROC curve that passes through the upper left corner (100% sensitivity, 100% specificity). Therefore the closer the ROC curve is to the upper left corner, the higher the overall accuracy of the test. Usually, ROC area higher (closer) to 1 is considered good, and closer to 0.0 is considered poor.

6.13 Result

The implementation involves employment of several software such as; WEKA version 3.6.4 software for performing the classification, and MatLab for feature selection. Table 6.17 shows the effectiveness of different data mining approaches. We had applied k cross validation, with k={2,3,4,5,6,7,8,9,10} for each of the data mining algorithms, and we observed that with k = 10 most of the algorithms provided the best accuracy. By comparing the evaluation measures achieved by each of the data mining techniques, we observe that SVM - Normalized PolyKernel has performed the best, and NN-Backpropagation exhibited the worst results. This could be attributed to the fact that and

NN-Backpropagation follows a heuristic path and usually converges only to locally optimal solutions and can suffer from multiple local minima, while SVM - Normalized PolyKernel always finds a unique global minimum. Through our experimental analysis we found that SVM-Normalized Polynomial Kernel provided an average of 98.5% true positive rate. With 99% true detection rate of malware as malware, the average weight for the false alarm rate achieved was about 2% in this case. Overall, SVM-Normalized Polynomial Kernel had outperformed all other classification methods in all measures, namely, TP Rate, FP Rate, Precision, Recall, F-Measure and ROC Area.

Table 6.17 Results Nine Classifiers at k = 10

		TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
J48		0.919	0.057	0.947	0.919	0.933	0.931	Malware
		0.943	0.081	0.913	0.943	0.928	0.931	Benign
	Weighted Avg.	**0.93**	**0.068**	**0.931**	**0.93**	**0.93**	**0.931**	
KNN		0.938	0.041	0.962	0.938	0.95	0.966	Malware
		0.959	0.062	0.933	0.959	0.946	0.966	Benign
	Weighted Avg.	**0.948**	**0.051**	**0.948**	**0.948**	**0.948**	**0.966**	
NB		0.913	0.094	0.915	0.913	0.914	0.94	Malware
		0.906	0.087	0.904	0.906	0.905	0.936	Benign
	Weighted Avg.	**0.91**	**0.09**	**0.91**	**0.91**	**0.91**	**0.938**	
NN - BackPropagation		0.983	0.301	0.82	0.983	0.894	0.744	Malware
		0.699	0.017	0.966	0.699	0.811	0.839	Benign
	Weighted Avg.	**0.864**	**0.183**	**0.881**	**0.864**	**0.859**	**0.783**	
Simple Logistic		0.931	0.07	0.936	0.931	0.934	0.97	Malware
		0.93	0.069	0.924	0.93	0.927	0.97	Benign
	Weighted Avg.	**0.93**	**0.07**	**0.93**	**0.93**	**0.93**	**0.97**	
SVM - Normalized PolyKernel		0.99	0.018	0.982	0.99	0.986	0.981	Malware
		0.981	0.031	0.969	0.981	0.983	0.982	Benign
	Weighted Avg.	**0.986**	**0.025**	**0.976**	**0.986**	**0.984**	**0.982**	
SVM - PolyKernel		0.966	0.102	0.913	0.966	0.939	0.932	Malware
		0.898	0.034	0.96	0.898	0.928	0.932	Benign
	Weighted Avg.	**0.934**	**0.069**	**0.936**	**0.934**	**0.934**	**0.932**	
SVM – Puk		0.901	0.033	0.968	0.901	0.933	0.934	Malware
		0.967	0.099	0.898	0.967	0.931	0.934	Benign
	Weighted Avg.	**0.932**	**0.064**	**0.935**	**0.932**	**0.932**	**0.934**	
SVM- Radial Basis Function (RBF)		0.94	0.084	0.925	0.94	0.933	0.928	Malware
		0.916	0.06	0.932	0.916	0.924	0.928	Benign
	Weighted Avg.	**0.929**	**0.073**	**0.929**	**0.929**	**0.929**	**0.928**	

Chapter 7 : Conclusions

"Pay no attention to the man behind the curtain."

— The Wizard of Oz

7.1 Overview

The final chapter of the dissertation provides an overall summary of this research study and briefly describes the proposed techniques, the results achieved, and the conclusions arrived at. The first section will be an open discussion on the state-of-the-art of this research topic. The next sections and subsections will summarise the various malware detection techniques adopted and their contribution to the information security field of knowledge. The last section throws some light on possible future research directions, along with final thoughts.

7.2 Discussion

Malware contains code that is designed to perform illegal activities, to cause damage, and to affect the integrity and the functionality of the digital system. This thesis has proposed and evaluated novel techniques to automatically detect hidden and obfuscated malware, aiming to address the malware threats by proposing a variety of novel digital forensic and data mining techniques. These techniques take a step to fill the lacuna found in literature to detect zero-day malware effectively due to the recent obfuscation strategies adopted by malware writers.

Malware attackers are taking advantage of our increased reliance on digital systems. As the number of cybercrime and computer attacks have increased exponentially, there is a need for developing standards in evidence collection and conducting malware analysis effectively as part of both the incident response and forensic analysis processes. The continued growth and diversification of the Internet has resulted in the increasing sophistication of tools and methods used to conduct computer system attacks and intrusions, these attacks can take in physical or logical places of a computer. Among these attacks, zero-day malware forms the biggest threat to information security. With more and more use of computers, portable devices and the Internet in everyday life, identification of new or unknown malware has become a major challenge in digital forensics and computer security. Hence, this research has focused on detecting such malware that lie hidden in the physical and logical space of a computer, evading from anti-virus scanners.

7.2.1 State-Of-The-Art

Cyber criminals are leveraging innovation at a fast pace to target many organizations, and security vendors cannot match this pace (Stabek *et al.* 2010). Effective deterrents to cybercrime are not known, available, or accessible to many practitioners, many of whom underestimate the scope and severity of the problem (McCombie *et al.* 2009; Watters 2009). In our view the key for fast speed in malware growth is the lack of understanding of the various types of hidden malware and their capabilities to exploit file system vulnerabilities. Security breaches are increasing in frequency and sophistication. This thesis presented along all the chapters details have been presented for the

abovementioned attacking trend with a view to understand the various behaviour of hidden malicious code that could be categorized as distinct malware types. Organizations should understand how they are viewed by cyber criminals in terms of attack vectors, systems of interest, and process vulnerabilities, so that they can better protect themselves from the Zeus generations and other attacks.

The normalization of different variants of malware to a single normal form could be effective, but studies indicate that they do not always lead to convergence. Certain heuristic methods also result in high false positives, thereby warranting the need for new methods that leverage on the knowledge that could be gained from the anomalies of these obfuscated malware.

7.2.2 Proposed Detection Methods

This dissertation focuses on the research problem of detection of hidden and obfuscated malware, which is generally considered as the first step in the malware defense. With proper identification of malware, it is possible to defend against infection. Unfortunately, there are multiple reasons why it is unlikely that one identification method will be universally effective. Hence, in this dissertation, multiple research problems related to the infection strategies of malware authors have been studied in Chapter 1. These strategies attempt to bypass the most popular malware detection method, which is based on fixed code signatures. In this research, five broadly classified methods that have adopted the potential techniques to counter such obfuscated malware strategies have been proposed. They are:

1. Forensic analysis of the NTFS – to detect hidden malware in NTFS file system slack space partitions,
2. Extract features out of executable binaries automatically to perform statistical differentiation of the op-code and API calls frequencies of malware and benign programs.
3. OP-code based detection – to detect anomalies and classify unknown malware based on the extended x86 IA-32 binary assembly instructions' frequency statistics, using i) Maximum Relevance (MR) filter heuristic, ii) Artificial Neural Net Input Gain Measurement Approximation (ANNIGMA), and iii) combination of MR and ANNIGMA.
4. API calls similarity based detection of unknown and obfuscated malware using various distance measures of vector models to detect obfuscated malware families.
5. API call based detection- to effectively detect and classify malware using ten robust supervised machine learning algorithms on API function call features.

All the above mentioned methods are fully automated and work on binary executables to detect hidden and obfuscated malware effectively as well as efficiently to address the zero-day malware problem.

7.3 Forensic Analysis of the NTFS

This research investigated the offline NTFS file systems and proposed effective digital forensic techniques that could be used to analyse and acquire evidences of hidden malware in NTFS disk images. An innovative methodology was proposed and evaluated to effectively detect hidden malware in physical space of a computer. Since malware

attackers take advantage of the NTFS weaknesses and the inability of existing Anti-virus to check the slack space, a guideline to investigate slack space for identifying known and unknown malware forms a significant contribution of the study.

Recent methods adopted by computer intruders, attackers and malware are to target hidden and deleted data so that they could evade from virus scanners and become extremely difficult to be identified using existing digital forensic tools. This work has attempted to explore the difficulties involved in digital forensics, especially in conducting NTFS disk image analysis and to propose digital forensic analysis procedure or steps to identify certain hidden malware effectively, while they reside in the system invisible to popular anti-virus scanners.

Through this empirical study, it is observed that the boot sector of the NTFS file system could be used as a vehicle to hide data by computer attackers as there is a potential file system weakness. The knowledge of file systems is important for digital forensics, as several techniques to hide data such as slack space and hidden attributes are being recently adopted by attackers. This is a potential NTFS file system weakness to be addressed and research in this domain area could lead to effective solution for the open problem of detecting new malicious codes that make use of such an obfuscated mode of attack. The results of the experiments conducted in this work show that the existing forensic software tools are not competent enough to comprehensively detect all hidden data in boot sectors.

As a first step to address this problem, a three-stage forensic analysis process consisting of nine steps is proposed to facilitate the experimental study. The results

gathered by following these proposed steps are reported. By adopting such a comprehensive process, this research could successfully identify all the unknown hidden malware in the $Boot file that had previously escaped undetected when existing forensic tools were used.

This pilot study has adopted a few forensic techniques and effective manual inspections of the NTFS file image. With these observations an automated tool based on the proposed process was developed to facilitate forensic analysis of the NTFS disk image in an efficient and comprehensive manner. Future plan is to extract and extrapolate malware signatures effectively as well as intelligently for any existing and even new malware that use hidden and obfuscated modes of attack.

Verification and Validation: The utilities described above to perform the forensic investigation steps have built-in MD5 hashing features. In digital forensic analysis, using a hashing technique is important to ensure data integrity and to identify which values of data have been maliciously changed as well as to explore known data objects. In addition, the 'Check the Data Integrity' step verifies the integrity of data again for test of congruence. Hence, verification and validation method is a simple process of checking the presence or absence of data in slack space, which determines the presence or absence of hidden malware correctly.

7.4 OP-Code Based Detection

Malware that make use of obfuscation of extended x86 IA-32 operation codes (op-codes) pose a great challenge for malware detectors as they can easily evade current signature-

based as well as heuristic-based detection engines. A novel approach has been proposed that combines op-code frequency statistics and hybrid wrapper-filter based feature selection technique in order to design a malware detector for identifying the anomalies of malware infecting the logical space of a computer.

With evasion techniques such as packers, polymorphism and metamorphism, recent malware is able to defeat current signature based detection techniques. In this chapter, a novel signature-free method for the detection of such obfuscated malware was proposed. Instead of using signature, we have used frequency statistics of op-codes with a wrapper-induction algorithm. One of the main contributions of this research is the development of a fully-automated algorithm to unpack, de-obfuscate and reverse engineer the binary executable without any need for manual inspection of assembly codes, and thereby we are able to find the op-code frequency statistics without any manual intervention.

To compute the frequency statistics for all op-codes each time for each executable during the scanning process is a difficult task. Hence, a novel hybrid wrapper-filter based signature-free approach to select the most important op-codes for this malware detection task is used. The novelty of our approach is that this integrates knowledge (from the intrinsic characteristics of data) obtained by the filter approach into the wrapper approach and combines the wrapper's heuristic score with the filter's ranking score in the wrapper stage of the hybrid. The main novelty is that this approach is signature-free and is able detect the malware variants that are very hard to defect with signature-based approaches. The combined heuristics in the hybrid (Maximum Relevance (MR) and Artificial Neural Network (ANN) based wrapper approach, where Artificial Neural Network Input Gain

Measurement Approximation (ANNIGMA)) (MR-ANNIGMA) takes the advantages of the complementary properties of the both filter and wrapper heuristics helps to guide the wrapper to find optimal and compact op-code subsets. Experimental results on real world malware and benign datasets show that our frequency statistics based approach achieves an accuracy of 97.529% with a very compact set of op-codes. Some limitations of the proposed approach could be avoided by applying other wrapper approaches and rule-generation processes.

7.5 API Feature Based Detection

The op-code base detection approach for malware detection exhibited good performance in detecting unkown malware. However, the behaviour patterns could not be studied. Hence, the next research step involved extracting behaviour patterns for detecting anomalies of such zero-day malware infecting the executables in the logical space of the computer. It is a common practice to undergo manual unpacking or static unpacking of executables using existing software tools, and to use human expertise in analysing the application programming interface calls for malware detection. However, extracting these features from the unpacked executables for reverse obfuscation is time consuming, labour intensive, and requires deep understanding of kernel and low-level programming such as assembly programming. Thus, security researchers and the anti-virus industry are facing a herculean task in extracting payloads hidden within packed executables with much human intervention.

7.5.1 API Behaviour Analysis

A statistical analysis of the Windows API calling sequence reflects the behaviour of a particular piece of code. In this research project, the API calls from the binary of a program are extracted to analyse the most common malware behaviour patterns and to classify program executables as malicious or benign. The extracted calls are subjected to a statistical test to determine the malware class based on suspicious behaviour. The entire static detection process was a fully-automated system and a four-step methodology was adopted for developing the system. Experimental tests were conducted using 21942 samples of malware and arrived at six main categories of suspicious behaviour of API call features. These being i) Search files, ii) Copy/Delete files, iii) Get file information, iv) Move Files, v) Read /Write files and vi) Change file attributes. Among these, API calls for Read /Write files were predominantly used by malware as a vehicle to infect the program.

In the research, the behavioural and structural features based on API calls are automatically extracted from the binary of a program. The extracted features are subjected to a statistical n-gram analysis to classify a program as either malicious or benign effectively with the aid of a supervised SVM machine learning technique. In the field of analysis the behavious of API finction calls four regions we covered. The first and foremost one is, outlining of a methodology to extract behaviour features of API calls that relate to various malware behaviour such as i) hooking of the system services, ii) creating or modifying files, iii) getting information from the file for making changes about the DLLs loaded by the malware. Second is providing a statistical analysis of the

API calls from the programs using n-gram model. The n-gram analyses the similarities and the distance of unknown malware with known behaviour so that obfuscated malware could be detected efficiently. Third developing a fully-automated tool to unpack, de-obfuscate and reverse engineer the program codes without any need for manual inspection of assembly codes. The last one is, applying SVM machine learning to train the classifier for a robust identification of known as well as unknown malware. Our experiments have shown the initial experimental result of 96.5% accuracy for unigrams is still very promising as a benchmark for improvements in our future research work.

7.5.2 API Similarity based detection

The proposed similarity based detection of unknown and obfuscated malware using API call features as an effective method in classifying and identifying zero day malware with existing malware families. This dissertation has proposed an approach to use frequency of occurrence of each Windows Application Programming Interface (API) calls with similarity mining analysis to detect new malware based on the similarities of distance measures. Different distance measures were implemented and similarity analysis were performed by using eight commonly adopted distance measures in vector models, namely Cosine, Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Euclidean, and Hamming distance similarity measure for Nearest Neighbor (NN).

The experimental investigations of the similarity analysis on large datasets of recent unknown malware and benign executables conclude that malware exhibit much similarity of API call features that can be used to classify them into their families effectively, and they are very much dissimilar from benign datasets. Also, the study reveals that there is

no similarity among benign files resulting in accurate identification of all benign files. Our fully automated system implementing the proposed approach is fast and effective and uses several similarity measures.

The experimental result of the similarity analysis aided in the malware classification and was evaluated using very large real-life malware dataset. The Similarity distance system used on our experiment was able to automatically identify all malware variants. We showed how to detect as a variant of the original malware.

In order to provide an accurate result of the we have taken the mean values for 8 different similarity measurements applied to the entire dataset, when the threshold for the similarity ratio is less than or equal to 0.5. The overall results demonstrate that there is low distance between malware variants but not with the benign programs. Also, the results showed there is high distance between benign dataset.

Similarity analysis is very efficient and effective to detect Malware variants from the same family or different families of Malware. Also, the experiments confirm that there is no similarity among the benign files, which is logical and true. Another important observation is that it is very hard to find similarity between Malware dataset and benign dataset, thereby validating that the proposed system is able to clearly distinguish between malware and benign datasets. In conclusion, Malware can be classified according to similarity and further, similarly test can be applied to detect malware variants. The proposed similarity based detection and classification of malware is further improved in performance by using feature selection, especially when large datasets require more computing memory and time for processing the extracted features.

7.5.3 Data Mining of API Call Features

Countermeasures such as antivirus detectors are unable to detect new malware and are in search of employing effective techniques, since the latest new malware adopt obfuscations to evade detection. With an exponential growth in unknown malware arising from innumerable automated obfuscations, there is a need to establish malware detection methods that are robust and efficient. In this thesis, we have proposed and developed a machine learning framework using eight different classifiers to detect unknown malware and to achieve high accuracy rate. In this work, iterative patterns based on Windows API calls have been used and statistical measures have been adopted to further improve the classification results. Our experiments conducted on large malware datasets have shown very promising results achieving more than 98.5% accuracy rate.

The main objective of the research reported was to propose and develop a machine learning framework to detect and classify unknown malware and to achieve high accuracy rate and low false alarm rate. Signature based detection method used by countermeasure is not enough to get an acceptable protection against the new malware. Malware detection by using machine learning is very much required by anti-virus vendors to detect unknown malware along with signature based detection for detecting existing malware. Since most Anti-Virus engines manage to have a detection rate of over 90% (Gavrilut *et al.* 2009), the detection framework proposed to detect zero-day attack is a very significant contribution. In this work, iterative patterns based on Windows API calls have been used and statistical measures to further improve the classification results. Overall, the salient achievements of the research reported in this chapter are:

- The proposed machine learning framework has resulted in high accuracies in malware detection. This is attributed to the unique feature selection of API sequences and the development of a fully-automated system used for evaluating data mining algorithms on large datasets of unknown malware.

- The proposed system is efficient as it uses filter approaches to be able to successfully detect malware with a smaller feature set. The term frequency of reduced API feature set using SVM (normalised poly kernel) has performed the best among the nine classifiers evaluated in this study.
- The system is signature-free and does not require knowledge or detailed study about the API sequence of execution to classify a malware.

Code obfuscation techniques can modify the parent code to produce offspring copies which have the same functionality but with different signatures to infect the logical space of a computer in order to evade the 'Signature-based' detection process easily. Since there are many to generate new obfuscated code, new generations of evaluated signatures are growing and the level of sophistication of unknown malware is also increasing. However, data mining of the offspring copies through pattern recognition can detect the obfuscated code. Hence, the in the final step of this research, data mining approach was adopted for malware and benign classification.

Also, a data mining approach within a machine learning framework was proposed to detect malicious programs, to learn from the behaviour of existing malicious and benign database of very large data sets of obfuscated unknown malware and benign programs. Supervised learning was performed using a dataset to train, validate and test,

an array of classifiers. The robust classifiers adopted were; The Naive Bayes (NB) Algorithm, The k−Nearest Neighbor (kNN) Algorithm, The Sequential Minimal Optimization (SMO) Algorithm with 4 differents kernels (SMO - Normalized PolyKernel, SMO – PolyKernel, SMO – Puk, and SMO- Radial Basis Function (RBF)), Backpropagation Neural Networks Algorithm, Logistic Regression, and J48 decision tree. Experiments conducted on large malware datasets resulted in 98.5% accuracy rate of malware detection.

7.6 Future Work and Final Thoughts

As future work, the following are the intended research possibilities:

NTFS File System versions: The investigations on malware infection in physical space described in Chapter 3 has focused on NTFS file system While Windows XP and Windows Server 2003 use the same NTFS version, Windows Vista uses the NTFS 3.1, and currently there is NTFS 4.0 and NTFS 5.0 released in the market. Therefore, malware detection in the physical space of the NTFS file system partition should cater to all the NTFS versions.

Combine Features: Chapters 4, Chapter 5, and Chapter 6 Chapter 6 proposed signature-free detection based on op-code and API calls, independently and future investigation would be to combine the classifiers intelligently. Combining the output of different classifiers instead of choosing the best one among them could be investigated in terms of detection accuracy and ROC curve. Moreover, the classifiers could be trained on different

datasets and to combine the classifiers learned on op-code and API calls efficiently and the final class output could be assigned using a voting strategy.

Time Complexity: Further improvements for real-time implementation of the fully-automated malware detection systems would include integrating both physical space forensic and logical space analysis, and taking into account the space-time complexity of the various algorithms implemented in this research work.

New Methods to Improve Detection: It is imperative to keep employing new techniques for studying the new polymorphic and metamorphic malware attacks in order to counter them. There are ways that the proposed method could be bypassed such as using lengthy loops or using run-time environment parameters in a polymorphic exploit. However, this requires the attackers to carefully craft their exploit code. In such a case, it would be worth further study of new techniques to detect these polymorphic exploits and the focus would be on generalizing the method for less obvious sequences of byte decoding.

New Approaches to Enhance the Accuracy: In future studies on completely new techniques or improving the proposed techniques to enhance the accuracy of static analysis would be explored: Static analysis is the basis of the three works proposed in this dissertation. In particular, it is the key to successfully characterizing program behaviours for metamorphic malware detection. Therefore, it is important to enhance its accuracy further to achieve near 100%. Through the investigations conducted in this research work, several ideas have been generated to improve the proposed static analysis approach that characterizes program behaviours based on system calls. These new ideas could be explored further.

Combined Static and Dynamic Analysis: Combining localized dynamic analysis with static analysis would be another prospective research worth exploring. Such investigations could be used to detect live malware targeting web. Many techniques can thwart static analysis such as using self-modifying code or indirect control transfer instructions. Making obfuscated system calls is a technique that specifically bypasses the static analysis approach. Using dynamic analysis techniques can address these problems. For instance, techniques could be designed to “locally" emulate instruction execution to deal with self-contained behaviours such as self-modifying and the obfuscated system calls. Static analysis could then be used to develop information to determine a “local" scope for a round of emulated instruction execution.

Modeling System Calls for Intrusion Detection: Another improvement in the proposed approach of API call feature extraction is by incorporating more types of system call parameters into pattern generation: This will improve the accuracy of pattern generation and similarity analysis. Besides the target address of a system call, the function argument(s) of a system call could also be used in pattern generation. The information on what function argument(s) a system call is using can possibly be obtained in two ways: 1) from system call specifications; 2) from instruction compilation heuristics: a parameter is usually passed through specific registers, i.e., eax.

Characterize the Malicious Behavior of Applications: Another topic of interest would be to investigate new approaches that characterize malicious program behaviours beside the system call based ones. For instance, looping structure for processing various control and commands in malicious bot binaries is a characteristic behaviour of bot software.

Combining Classifiers: In Chapters 4, 5, and 6 different classifiers were discussed and compared, the results identified the best was SVM - Normalized Polynomial kernel, based on classification accuracy and ROC curves. Future work could combine classifiers as combining classifiers have given excellent results in other areas of application. Combining the output of different classifiers instead of choosing the best one among them could be investigated to achieve better accuracy. Also, in the current research, classifiers were trained on the same dataset and future work would experiment on training with different datasets. For different datasets, to combine the classifiers learned on op-code features and API call features would be explored. In the case of training the same dataset, several classifiers could be employed to result in a final class output assigned by adopting a voting strategy.

More Classifiers: The machine learning data mining framework proposed in this research could include more classification techniques. Several other classifiers could be added to this list including random forest and bagging in future research.

Time and Space Complexity: The major limitation for real time implementation of such a data mining framework in malware detection is the time and space complexity of the machine learning and other algorithms involved. The disassembly, parsing and feature extraction are time consuming processes and require a lot of memory space as well. In an automated framework these processes need to be streamlined for better efficiency.

Real World Implementation: This work was focused on academic research and the implementation of the proposed data mining framework for a real world malware detection system that could be used commercially was not discussed.

Real World Commercialisation: This work has an academic research focus and hence the commercialisation of the proposed malware detection methods and implementation has not been discussed here.

Abbreviations

AIC	Information Criterion
AIS	Artificial Immune System
API	Application Programming Interfaces
ATA/IDE	AT Attachment interface
AV	Anti-Virus
BE	Backward Elimination
CFG	Control Flow Graph
CFG	Control Flow Graph Personal Computer
CFG	Control Flow Graph
COFF	Common Object File Format
COM	Command file
CPL	Control Panel Applets
CPU	Control Processing Unit
CRC	Cyclic Redundancy Check
DLL	Dynamic Link Library
DOS	Disk Operating System
EIDIP	Enhanced Digital Investigation Process
FAT	File Allocation Table
FN	False Negative
FP	False Positive
GLM	generalized linear models

HMM	Hidden Markov Model
IACIS	International Association of Computer Investigation Specialists
IAT	Import Address Table
IDA Pro	Interactive Disassembler Pro
IDIP	Integrated Digital Investigation Model
KB	Kilobyte
kNN	k-Nearest Neighbor
LMSW	Load Machine Status Word
MC	Malicious Code
MFT	Master File Table
MI	Mutual Information
MPCGEN	Mass Code Generator
MR	Maximum Relevance
NB	Naive Bayes
NGVCK	Next Generation Virus Creation Kit
NIJ	National Institute of Justice
NN	Neural Network
NOP	No Operation Performed
NTFS	Windows NT File System
OEP	Original Entry Point
OS	Operating System
PC	Personal Computer

PCA	principle component analysis
PDF	Portable Document Format
PDG	Program Dependence Graph
PE	Portable Executable
PS-MPC	Phalcon/Skism Mass Produced Code Generator
PUI	Process Under Inspection
RAM	Random Access Memory
RCFL	Regional Computer Forensics Laboratory
ROC	Receiver Operating Characteristic
SCR	Screensavers
SCSI	Small Computer System Interface
SDSC	San Diego Supercomputer Center
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
SWGDE	Scientific Working Group on Digital Evidence
TB	Terabyte
TLS	Thread Local Storage
TN	True Negative
TP	True Positive
VM	Virtual Machine

Bibliography

Adankon, M. & Cheriet, M. (2011). 'Help-Training for semi-supervised support vector machines', *Computer Analysis of Images and Patterns - Pattern Recognition*, vol. 44, no. 9, pp. 2220–2230.

Ahmad, A. (2002). 'The Forensic Chain-of-Evidence Model: Improving the Process of Evidence Collection in Incident Handling Procedures', *The 6th Pacific Asia Conference on Information Systems*: Citeseer, Tokyo, Japan.

Alazab, M. (2009). 'Effective forensic techniques for static analysis of NTFS file system analysis', *Annual Research Conference*: 7 Nov, Ballarat, VIC, p. 23.

Alazab, M. (2010). 'Static Analysis of Obfuscated Malware', *Annual Research Conference*: University of Ballarat, 14 Nov, Ballarat, VIC, p. 17.

Alazab, M. (2011). 'Static analysis for Anomaly and Similarity based detection', *Annual Research Conference*: University of Ballarat, 4 Nov, Ballarat, VIC, p. 87.

Alazab, M., Layton, R., Venkataraman, S. & Watters, P. (2010). 'Malware Detection Based on Structural and Behavioural Features of API calls', *The 1st International Cyber Resilience Conference*: Security Research Centre, Edith Cowan University, Perth, Western Australia, pp. 1-10.

Alazab, M., Venkataraman, S. & Watters, P. (2009). 'Effective digital forensic analysis of the NTFS disk image', *Ubiquitous Computing and Communication Journal*, vol. 4, no. 1, pp. 551- 558.

Alazab, M., Venkataraman, S. & Watters, P. (2010). 'Towards Understanding Malware Behaviour by the Extraction of API Calls', *Cybercrime and Trustworthy Computing Workshop*: IEEE Computer Society, 19-20 July, Ballarat, VIC, pp. 52-59.

Alazab, M., Venkatraman, S. & Watters, P. (2009). 'Digital forensic techniques for static analysis of NTFS images', *The 4th International Conference on Information Technology*: IEEE Computer Society, 3- 5 Jun, Amman- Jordan, pp. 1- 9.

Alazab, M., Venkatraman, S., Watters, P. & Alazab, M. (2011). 'Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures', P. Vamplew, A. Stranieri, K.-L. Ong, P. Christen & P. Kennedy (eds), *The 9th Australasian Data Mining Conference*: Australian Computer Society, 1- 2 Dec, Ballarat, VIC, vol. 121.

Alazab, M., Watters, P., Venkatraman, S., Alazab, A. & Alazab, M. (2011). 'Cybercrime: Current Trends of Malware Threats', *The International Conference in Global Security*

Safety and Sustainability / International Conference on e-Democracy: Springer, Thessaloniki, Greece, pp. 1- 7.

Aleskerov, E., Freisleben, B. & Rao, B. (1997). 'CARDWATCH: a neural network based database mining system for credit card fraud detection', *The IEEE/IAFE Computational Intelligence for Financial Engineering*: IEEE Computer Society, 23-25 Mar, New York, USA pp. 220-226.

Alperovitch, D. D., T.; Greve, P.; Kashyap, R.; Marcus, D.; Masiello, S.; Paget, F. & Schmugar, C. (2011). 'McAfee Labs - 2011 Threats Predictions', *McAfee, Inc.*

Alsagoff, S. (2011). 'Manual Removal of Malware – Is It Still Relevant?', *International Journal of Research and Reviews in Information Security and Privacy*, vol. 1, no. 1.

Altunaya, M., Leyfferb, S., Linderothc, J. & Xieb, Z. (2011). 'Optimal response to attacks on the open science grid', *Computer Networks*, vol. 55, no. 1, pp. 61-73.

Andrew, M. (2007). 'Defining a Process Model for Forensic Analysis of Digital Devices and Storage Media', *Systematic Approaches to Digital Forensic Engineering*: IEEE Computer Society, 10-12 Apr, Bell Harbor, WA, pp. 16 - 30.

Attaluri, S. (2007). 'Detecting metamorphic viruses using profile Hidden Markov Models', Master Degree thesis, San Jose State University.

Attaluri, S. & McGhee, S. (2009). 'Profile hidden Markov models and metamorphic virus detection', *Journal in Computer Virology*, vol. 5, no. 2, pp. 151-169.

Aycock, J. (2006). *Computer Viruses and Malware*, vol. 22, Advances in Information Security, Springer.

Bakshi, A., Dixit, V. & Mehta, K. (2010). 'Virus: A Menace for Information Security', *Global Journal of Enterprise Information System*, vol. 2, no. 1, pp. 58-70.

Balagani, K. S. & Phoha, V. V. (2010). 'On the Feature Selection Criterion Based on an Approximation of Multidimensional Mutual Information', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1342-1343.

Balakrishnan, A. & Schulze, C. (2005). 'Code Obfuscation Literature Survey', vol. 19, pp. 1-10.

Ban, T., Ando, R. & Kadobayashi, Y. (2010). 'A Fast Kernel on Hierarchial Tree Structures and Its Application to Windows Application Behavior Analysis', in K. Wong, B. Mendis & A. Bouzerdoum (eds), *Neural Information Processing. Models and Applications*, Springer Berlin / Heidelberg, vol. 6444, pp. 267-274.

Baryamureeba, V. & Tushabe, F. (2006). 'The Enhanced Digital Investigation Process Model', *Asian Journal of Information Technology*, vol. 5, no. 7, pp. 790-794.

Bhattacharyya, S., Jha, S., Tharakunnel, K. & Westland, J. C. (2011). 'Data mining for credit card fraud: A comparative study', *Decision Support Systems*, vol. 50, no. 3, pp. 602-613.

Bilar, D. (2007). 'Opcodes as predictor for malware', *Int. J. Electron. Secur. Digit. Forensic*, vol. 1, no. 2, pp. 156-168.

Birrer, B., Raines, R., Baldwin, R., Oxley, M. & Rogers, S. (2009). 'Using Qualia and Hierarchical Models in Malware Detection', *Special Issue on Intrusion and Malware Detection: Journal of Information Assurance and Security*, vol. 4, no. 3.

Bitdefender Antivirus Technology (2010). *White paper 'Bitdefender Antivirus Technology'*, Bitdefender Antivirus Technology. Retrieved from <http://www.bitdefender.com/files/Main/file/BitDefender_Antivirus_Technology.pdf>.

Blum, A. L. & Langley, P. (1997). 'Selection of relevant features and examples in machine learning', *Artif. Intell.*, vol. 97, no. 1-2, pp. 245-271.

Brown, P., deSouza, P., Mercer, R., Pietra, V. & Lai, J. (1992). 'Class-based n-gram models of natural language', *Computational Linguistics*, vol. 18, no. 4, pp. 467-479.

Bruschi, D., Martignoni, L. & Monga, M. (2006). 'Detecting Self-mutating Malware Using Control-Flow Graph Matching', in R. Büschkes & P. Laskov (eds), *Detection of Intrusions and Malware & Vulnerability Assessment*, Springer Berlin / Heidelberg, vol. 4064, pp. 129-143.

Camastra, F., Ciaramella, A. & Staiano, A. (2011). 'Machine learning and soft computing for ICT security: an overview of current trends', *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-13.

Carrie, B. (2003). 'Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layer', *International Journal of Digital Evidence*, vol. 1, no. 4, pp. 1-12.

Carrier, B. (2005). *File System Forensic Analysis*, 1st edn, Addison-Wesley Professional.

The Autopsy Forensic (2010). 'The Autopsy Forensic', version: 2.24. Retrieved from.

The Sleuth Kit (TSK) (2011). 'The Sleuth Kit (TSK)', version: 3.2.2. Retrieved from.

Carrier, B. & Spafford, E. (2003). 'Getting Physical with the Digital Investigation Process', *International Journal of Digital Evidence*, vol. 2, no. 2, pp. 1-20.

Casey, E. (2004). *Digital evidence and computer crime: forensic science, computers and the Internet*, 2nd edn, Academic Press, London.

Cesare, S. & Xiang, Y. (2010). 'Classification of Malware Using Structured Control Flow', *The 8th Australasian Symposium on Parallel and Distributed Computing* Australian Computer Society Inc, Brisbane, Australia, vol. 107, pp. 61-70.

Chan, K. Y. & Loh, W. Y. (2004). 'An Algorithm for Building Accurate and Comprehensible Logistic Regression Trees', *Journal of Computational and Graphical Statistics*, vol. 13, no. 4.

Chandola, V., Banerjee, A. & Kumar, V. (2009). 'Anomaly detection: A survey', *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 1-58.

LIBSVM - A Library for Support Vector Machines (2011). 'LIBSVM - A Library for Support Vector Machines', version: 3.1. Retrieved from.

Chang, H. & Atallah, M. (2002). 'Protecting Software Code by Guards', in T. Sander (ed.), *Security and Privacy in Digital Rights Management*, Springer Berlin / Heidelberg, vol. 2320, pp. 125-141.

Chen, W.-H., Hsu, S.-H. & Shen, H.-P. (2005). 'Application of SVM and ANN for intrusion detection', *Computers and Operations Research*, vol. 32, no. 10, pp. 2617-2634.

Choi, S., Park, H., Lim, H.-i. & Han, T. (2009). 'A static API birthmark for Windows binary executables', *Journal of Systems and Software*, vol. 82, no. 5, pp. 862-873.

Chouchane, M. R. & Lakhotia, A. (2006). 'Using engine signature to detect metamorphic malware', *The 4th ACM workshop on Recurring malcode*: ACM, Alexandria, Virginia, USA, pp. 73-78.

Christodorescu, M. & Jha, S. (2003). 'Static analysis of executables to detect malicious patterns', *The 12th conference on USENIX Security Symposium* USENIX Association, Washington, DC, vol. 12, pp. 12-12.

Christodorescu, M. & Jha, S. (2004). 'Testing Malware Detectors', *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 34-44.

Christodorescu, M., Jha, S., Seshia, S., Song, D. & Bryant, R. (2005). 'Semantics-Aware Malware Detection', *The IEEE Symposium on Security and Privacy*: IEEE Computer Society, 8-11 May, Oakland, USA, pp. 1-21.

Cifuentes, C. & Fraboulet, A. (1997). 'Intraprocedural static slicing of binary executables', *The 13th International Conference on Software Maintenance* IEEE Computer Society, Oct 1-3, Bari, Italy, pp. 188 - 195.

Cifuentes, C. & Gough, K. J. (1995). 'Decompilation of Binary Programs', *Software: Practice and Experience*, vol. 25, no. 7, pp. 811-829.

Cohen, F. (1987). 'Computer viruses:: Theory and experiments', *Computers & Security*, vol. 6, no. 1, pp. 22-35.

Cortes, C. & Vapnik, V. (1995). 'Support-Vector Networks', *Machine Learning*, vol. 20, no. 3, pp. 273-297.

Crescenzo, G. D. & Vakil, F. (2006). 'Cryptographic hashing for virus localization', *Proceedings of the 4th ACM workshop on Recurring malcode*: ACM, Alexandria, Virginia, USA, pp. 41-48.

Danny, B. (2010). 'Digging up the hacking underground', *Infosecurity*, vol. 7, no. 5, pp. 14-17.

Daoud, E. A., Jebril, I. & Zaqaibeh, B. (2008). 'Computer Virus Strategies and Detection Methods', *International Journal of Open Problems in Computer Science and Mathematics*, vol. 1, no. 2.

Dasgupta, D. (1997). 'Artificial neural networks and artificial immune systems: similarities and differences', *The IEEE International Conference on Systems, Man, and Cybernetics- Computational Cybernetics and Simulation*: IEEE Computer Society, 12-15 Oct, Orlando, FL , USA vol. 1, pp. 873-878

Dayhoff, J. & DeLeo, J. (2001). 'Artificial Neural Networks Opening the Black Box', *Cancer Supplement*, vol. 91, no. 8, pp. 1615-1635.

Desai, P. (2010). 'A highly metamorphic virus generator', *International Journal of Multimedia Intelligence and Security*, vol. 1, no. 4, pp. 402-427.

Dinaburg, A., Royal, P., Sharif, M. & Lee, W. (2008). 'Ether: malware analysis via hardware virtualization extensions', *Proceedings of the 15th ACM conference on Computer and communications security*: ACM, Alexandria, Virginia, USA, pp. 51-62.

Do, T. M. T. & Artières, T. (2009). 'Learning mixture models with support vector machines for sequence classification and segmentation', Pattern Recognition, vol. 42, no. 12, pp. 3224-3230.

Egele, M, Scholte, T, Kirda, E & Kruegel, C. (2012). 'A Survey on Automated Dynamic Malware Analysis Techniques and Tools', ACM Computing Surveys, vol. 44, no. 2, pp. 1-49.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering*, 1st edn, Wiley.

Eisner, A. (2003). *PSINet Europe Study Reveals Massive Vulnerabilities*, theWHIR.com, retrieved 3 May 2010. Retrieved from <http://www.thewhir.com/web-hosting-news/hackers>.

Elaiwat, S., Alazab, A., Venkatraman, S. & Alazab, M. (2010a). 'Applying Genetic Algorithm for Optimizing Broadcasting Process in Ad-hoc Network', *International Journal of Recent Trends in Engineering and Technology*, vol. 4, no. 1, pp. 68-72.

Elaiwat, S., Alazab, A., Venkatraman, S. & Alazab, M. (2010b). 'GOM: New genetic optimizing model for broadcasting tree in MANET', *The 2nd International Conference on Computer Technology and Development*: IEEE Computer Society, 2-4 Nov, Cairo, pp. 477-481.

Emigh, A. (2006). 'The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond', *Journal of Digital Forensic Practice*, vol. 1, no. 6.

Farmer, D. & Venema, W. (2005). *Forensic discovery*, vol. 6, Addison-Wesley Professional.

Ferrante, J., Ottenstein, K. J. & Warren, J. D. (1987). 'The program dependence graph and its use in optimization', *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319-349.

Ferrie, P. & Szor, P. (2001). 'Zmist opportunities', *Virus Bulletin*, pp. 6–7.

FICO (2010). *Fraud Predictor with Merchant Profiles* FICO, retrieved Junne 22 2010. Retrieved from <http://www.fico.com/en/Products/DMApps/Pages/Fraud-Predictor-with-Merchant-Profiles.aspx>.

Frawley, W., Piatetsky-shapiro, G. & Matheus, C. (1992). 'Knowledge discovery in databases: An overview', *Al Magazine*, vol. 13, no. 3, pp. 213–228.

Fukushima, Y., Sakai, A., Hori, Y. & Sakurai, K. (2010). 'A Behavior Based Malware Detection Scheme for Avoiding False Positive', *The 6th IEEE Workshop on Secure Network Protocols*: IEEE Computer Society, 5-5 Oct, Kyoto pp. 79 - 84

DD :Unix Command and Image Creation (1970). 'DD :Unix Command and Image Creation', version: 2.4.23. Retrieved from <http://www.softpanorama.org/Tools/dd.shtml>.

Gavrilut, D., Cimpoes, M., Anton, D. & Ciortuz, L. (2009). 'Malware Detection Using Machine Learning', *International Multiconference on Computer Science and Information Technology*: IEEE Computer Society, 12-14 Oct, Mragowo, Poland, pp. 735-741.

General Information Security Statistics (2004). *Security Stats*. Retrieved from <http://www.securitystats.com/infos-ec.html>.

Ghosh, S. & Turrini, E. (2010). *Cybercrimes: A Multidisciplinary Analysis*, Springer Verlag.

Govindaraju, A. (2010a). 'Although PHMM can detect malwares which are metamorphic', Master Degree thesis, San Jose State University.

Govindaraju, A. (2010b). 'Exhaustive Statistical Analysis for Detection of Metamorphic Malware', Master Degree thesis, San Jose State University.

Gu, G., Porras, P., Yegneswaran, V., Fong, M. & Lee, W. (2007). 'BotHunter: detecting malware infection through IDS-driven dialog correlation', *The 16th USENIX Security Symposium on USENIX Security Symposium*: USENIX Association, Boston, MA, pp. 1-16.

Guo, T. & Li, G.-Y. (2008). 'Neural data mining for credit card fraud detection', *Machine Learning and Cybernetics, International Conference on*: IEEE Computer Society, July 12-15 Kunming vol. 7, pp. 3630-3634.

Hand, D. J., Mannila, H. & Smyth, P. (2001). *Principles of data mining*, 1st edn, A Bradford Book.

DCFLDD: Enhanced version of GNU dd (2006). 'DCFLDD: Enhanced version of GNU dd ', version: 1.3.4-1. Retrieved from <http://dcfldd.sourceforge.net/>.

Hearst, M. A., Dumais, S. T., Osman, E., Platt, J. & Scholkopf, B. (1998). 'Support vector machines', *Intelligent Systems and their Applications, IEEE*, vol. 13, no. 4, pp. 18-28.

Holz, T., Marechal, S. & Raynal, F. (2006). 'New Threats and Attacks on the World Wide Web', *IEEE Security and Privacy, IEEE Educational Activities Department*, vol. 4, no. 2, pp. 72-75.

Hu, Q., Liu, J. & Yu, D. (2008). 'Mixed feature selection based on granulation and approximation', *Journal Knowledge-Based Systems*, vol. 21, no. 4, pp. 294-304.

Huda, S., Yearwood, J. & Strainieri, A. (2010). 'Hybrid wrapper-filter approaches for input feature selection using Maximum relevance and Artificial Neural Network Input Gain Measurement Approximation (ANNIGMA)', *Fourth International Conference on Network and System Security*: IEEE Computer Society, 1-3 Sep, Melbourne, Australia, pp. 442-449.

Huebner, E., Bem, D. & Wee, C. K. (2006). 'Data hiding in the NTFS file system', *Digital Investigation*, vol. 3, no. 4, pp. 211-226.

Hung-Ju, C.-N., Huang, H.-J. & Schuschel, D. (2002). 'The ANNIGMA-wrapper approach to fast feature selection for neural nets', *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, pp. 207-212.

IBM (2008). 'Improving Payments Fraud Detection and Prevention: ACI Proactive Risk Manager with IBM System z10 '.

IDA Pro Disassembler and Debugger (2010). 'IDA Pro Disassembler and Debugger ', version: 5.7. Retrieved from <http://www.hex-rays.com/idapro/overview.htm>.

IDAPython : Python Plugin for Interactive Disassembler Pro (2011). 'IDAPython : Python Plugin for Interactive Disassembler Pro', version: 1.4.3. Retrieved from <http://code.google.com/p/idapython/>.

Intel (2010a). 'Intel ® 64 and IA-32 Architectures Software Developer's Manuals : Basic Architecture', vol. 1.

Intel (2010b). 'Intel ® 64 and IA-32 Architectures Software Developer's Manuals : Instruction Set Reference, N-Z', vol. 2B.

Islam, R., Tian, R., Batten, L. & Versteeg, S. (2010). 'Classification of Malware Based on String and Function Feature Selection', *Cybercrime and Trustworthy Computing Workshop*: IEEE Computer Society, 19-20 July, Ballarat, pp. 9-17.

Jacob, G., Debar, H. & Filiol, E. (2008). 'Behavioral detection of malware: from a survey towards an established taxonomy', *Journal in Computer Virology*, vol. 4, no. 3, pp. 251-266.

Jacob, G., Filiol, E. & Debar, H. (2009). 'Functional polymorphic engines: formalisation, implementation and use cases', *Journal in Computer Virology*, vol. 5, no. 3, pp. 247-261.

Jahankhani, H. & Al-Nemrat, A. (2008). 'Global E-Security', in H. Jahankhani, K. Revett & D. Palmer-Brown (eds), *Global E-Security: Communications in Computer and Information Science*, Springer Berlin Heidelberg, London, UK, vol. 12, pp. 3-9.

Jahankhani, H. & Al-Nemrat, A. (2010). 'Examination of Cyber-criminal Behaviour', *International Journal of Information Science and Management, Special Issue*, pp. 41 - 48

James, D. (2007). 'Internet Security – the Threats Are Very Real', *Educators' eZine*.

Järvelin, A., Järvelin, A. & Järvelin, K. (2007). 's-grams: Defining generalized n-grams for information retrieval', *Information Processing & Management*, vol. 43, no. 4, pp. 1005-1019.

John, G., Kohavi, R. & Pfleger, K. (1994). 'Irrelevant Features and the Subset Selection Problem', *Machine Learning: Proceedings of the Eleventh International*, pp. 121-129.

Kang, M. G., Poosankam, P. & Yin, H. (2007). 'Renovo: a hidden code extractor for packed executables', *Proceedings of the 2007 ACM workshop on Recurring malcode*: ACM, Alexandria, Virginia, USA, pp. 46-53.

Karim, M., Walenstein A., Lakhotia A. & Parida L. (2005). 'Malware phylogeny generation using permutations of code'. *Journal in Computer Virology*, vol.1, no.1, pp.13–23.

Keizer, G. (2007). *Symantec false positive cripples thousands of Chinese PCs*, retrieved 3 March 2010. Retrieved from <http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9019958&intsrc=hm_list.>.

Khan, A., Wiil, U. K. & Memon, N. (2010). 'Digital Forensics and Crime Investigation: Legal Issues in Prosecution at National Level', *The 5th IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*: IEEE Computer Society, 20-20 May, pp. 133-140.

Frhed (2009). 'Frhed', version: 1.7.1. Retrieved from <http://frhed.sourceforge.net/en/>.

Kohavi, R. & John, G. H. (1997). 'Wrappers for feature subset selection', *Artificial Intelligence - Special issue on relevance*, vol. 97, no. 1-2, pp. 273-324.

Kolter, J. Z. & Maloof, M. A. (2006). 'Learning to Detect and Classify Malicious Executables in the Wild', *Journal of Machine Learning Research*, vol. 7, pp. 2721-2744.

Komisarczuk, P. (2010). 'Who Are We Fighting? Dealing with threats is one thing, finding them is another', *ISNOW: The magazine of the BCS security forum* vol. 5, no. 1.

Kong, D., Jhi, Y.-C., Gong, T., Zhu, S., Liu, P. & Xi, H. (2010). 'SAS: Semantics Aware Signature Generation for Polymorphic Worm Detection Security and Privacy in Communication Networks', in S. Jajodia & J. Zhou (eds), Springer Berlin Heidelberg, vol. 50, pp. 1-19.

Konstantinou, E. & Wolthusen, S. (2008). *Metamorphic virus: Analysis and detection*, Royal Holloway, London.

Kou, Y., Lu, C.-T., Sirwongwattana, S. & Huang, Y.-P. (2004). 'Survey of fraud detection techniques', *The IEEE International Conference on Networking, Sensing and Control*: IEEE Computer Society, Taipei, Taiwan, vol. 2, pp. 749-754.

Krogh, A. (1998). 'An introduction to hidden Markov models for biological sequences', in *Computational Methods in Molecular Biology*, Elsevier Science, pp. 45-63.

Kruegel, C., Robertson, W., Valeur, F. & Vigna, G. (2004). 'Static Disassembly of Obfuscated Binaries', *The 13th USENIX Security Symposium* USENIX magazine, 11- 13 Aug, San Diego, CA, USA, pp. 255–270.

Kruse, W. & Heiser, J. (2001). *Computer Forensics: Incident Response Essentials*, 1st edn, Addison-Wesley Professional.

Kuncheva, L. I. (2006). 'On the optimality of NaIve Bayes with dependent binary features', *Pattern Recognition Letters*, vol. 27, no. 7, pp. 830-837.

Landwehr, C. E., Bull, A. R., McDermott, J. P. & Choi, W. S. (1994). 'A taxonomy of computer program security flaws', *ACM Computing Surveys (CSUR)*, vol. 26, no. 3, pp. 211-254.

Lawton, G. (2002). 'Virus Wars: Fewer Attacks, New Threats', *IEEE Computer Society*, vol. 35, no. 12, pp. 22 - 24.

Layton, R., Brown, S. & Watters, P. (2009). 'Using Differencing to Increase Distinctiveness for Phishing Website Clustering', *Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*: IEEE Computer Society, Jul 07-09, Brisbane, Australia pp. 488-492.

Li, X., Loh, P. & Tan, F. (2011). 'Mechanisms of Polymorphic and Metamorphic Viruses', *The Intelligence and Security Informatics Conference*: IEEE Computer Society, 12-14 Sep, Athens, pp. 149-154.

Linn, C. & Debray, S. (2003). 'Obfuscation of executable code to improve resistance to static disassembly', *10th ACM conference on Computer and communications security* ACM, Oct 27–31, Washington, DC, USA, pp. 290-299.

Lobo, D., Watters, P. & Wu, X. (2010a). 'Identifying Rootkit Infections Using Data Mining', *The International Conference on Information Science and Applications*: IEEE Computer Society, 21-23 April, Seoul, pp. 1 - 7.

Lobo, D., Watters, P. & Wu, X. (2010b). ' RBACS: Rootkit Behavioral Analysis and Classification System', *The International Conference on Knowledge Discovery and Data Mining*: IEEE Computer Society, 9-10 Jan, Ballarat, pp. 75-80.

Lobo, D., Watters, P., Wu, X. & Sun, L. (2010). 'Windows Rootkits: Attacks and Countermeasures', *Cybercrime and Trustworthy Computing Workshop*: IEEE Computer Society, 19-20 July, Ballarat, pp. 69-78.

MacNamara, S., Cunningham, P. & Byrne, J. (1998). 'Neural networks for language identification: a comparative study', *Information Processing & Management*, vol. 34, no. 4, pp. 395-403.

Malan, D. J. & Smith, M. D. (2005). 'Host-based detection of worms through peer-to-peer cooperation', *The ACM workshop on Rapid malcode*: ACM, Fairfax, VA, USA, pp. 72-80.

Manap, S. (2001). *Rootkit: Attacker undercover tools*, Retrieved from.

Maria, T. A. (2011). 'The Growing Global Threat of Cyber-crime given the Current Economic Crisis: A Study regarding Internet Malicious Activities in Romania', *Acta Universitatis Danubius Economica*, vol. 10, no. 1, pp. 179-190.

Martignoni, L., Christodorescu, M. & Jha, S. (2007). 'OmniUnpack: Fast, Generic, and Safe Unpacking of Malware', *The 23rd Annual Computer Security Applications Conference*: 10-14 Dec, pp. 431-441.

Marx, A. (2004). *Antivirus outbreak response testing and impact*, AV-test.org, retrieved March 15 2010. Retrieved from <http://www.virusbtn.com/conference/vb2004/abstracts/amarx.xml>.

McCombie, S., Pieprzyk, J. & Watters, P. (2009). 'Cybercrime Attribution: An Eastern European Case Stud', *The 7th Australian Digital Forensics Conference*: School of Computer and Information Science, Edith Cowan University, Perth, Western Australia, pp. 41-51.

McGraw, G. & Morrisett, G. (2000). 'Attacking Malicious Code: A Report to the Infosec Research Council', *IEEE Software*, vol. 17, no. 5, pp. 33-41.

MetaPHOR (2010). *W32.Simile*, Symantec Enterprise Security,. Retrieved from <http://www.symantec.com/security_response/writeup.jsp?docid=2002-030617-5423-99>.

Microsoft Developer Network (2011). *Windows API Functions*, Microsoft, retrieved 12/10 2010. Retrieved from <http://msdn.microsoft.com/en-us/>.

Microsoft WinDbg (2010). *Microsoft Windows SDK for Windows 7 and .NET Framework 4* Microsoft retrieved 5/2 2011. Retrieved from <http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx>.

Moshchuk, E., Bragin, T., Gribble, S. D. & Levy, H. M. (2006). 'A crawler-based study of spyware on the Web ', *The 13th Network and Distributed System Security Symposium*: February, pp. 17-33.

Naiqi, L., Zhongshan, W., Yujie, H. & QinKe (2008). 'Computer Forensics Research and Implementation Based on NTFS File System', *The International Colloquium on Computing, Communication, Control, and Management*: IEEE Computer Society, 3-4 Aug, Guangzhou, pp. 519 - 523.

Orr (2006). *The Viral Darwinism of W32.Evol An In-depth Analysis of a Metamorphic Engine*.

Orr (2007). *The Molecular Virology of Lexotan32 Metamorphism Illustrated*.

Palmer, G. (2001). *A Road Map for Digital Forensic Research*, Utica, New York.

Park, H., Choi, S., Lim, H.-i. & Han, T. (2008a). 'Detecting code theft via a static instruction trace birthmark for Java methods', *International Conference on Industrial Informatics*: IEEE Computer Society, JuL 13-16 Daejeon pp. 551-556.

Park, H., Choi, S., Lim, H.-i. & Han, T. (2008b). 'Detecting Java Theft Based on Static API Trace Birthmark', in K. Matsuura & E. Fujisaki (eds), *Advances in Information and Computer Security*, Springer Berlin / Heidelberg, vol. 5312, pp. 121-135.

Passerini, E., Paleari, R. & Martignoni, L. (2009). 'How Good Are Malware Detectors at Remediating Infected Systems?', in U. Flegel & D. Bruschi (eds), *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer Berlin / Heidelberg, vol. 5587, pp. 21-37.

Patcha, A. & Park, J.-M. (2007). 'An overview of anomaly detection techniques: Existing solutions and latest technological trends', *Computer Networks*, vol. 51, no. 12, pp. 3448-3470.

Paul, N. (2008). 'Disk-Level Behavioral Malware Detection', Doctor of Philosophy thesis, University of Virginia.

(2009). version: 1.99 R6. Retrieved from <http://www.heaventools.com/overview.htm>.

PEiD (2008). *Detect most common packers, cryptors and compilers for PE files*, peid.info, retrieved 02/06 2010. Retrieved from <http://www.peid.info/>.

Perriot, F. & Ferrie, P. (2004). 'Principles and practise of x-raying', *Virus Bulletin Conference* September, pp. 1- 17.

HexEdit (2010). 'HexEdit', version: 1.03. Retrieved from <http://www.hexedit.com/>.

Preda, M. D., Christodorescu, M., Jha, S. & Debray, S. (2008). 'A Semantics-Based Approach to Malware Detection', *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 30, no. 5, pp. 1-54.

Purcell, D. & Lang, S.-D. (2008). 'Forensic Artifacts of Microsoft Windows Vista System', in C. Yang, H. Chen, M. Chau, K. Chang, S.-D. Lang, P. Chen, R. Hsieh, D. Zeng, F.-Y. Wang, K. Carley, W. Mao & J. Zhan (eds), *Intelligence and Security Informatics*, Springer Berlin / Heidelberg, vol. 5075, pp. 304-319.

Rabek, J. C., Khazan, R. I., Lewandowski, S. M. & Cunningham, R. K. (2003). 'Detection of injected, dynamically generated, and obfuscated malicious code', *Proceedings of the 2003 ACM workshop on Rapid malcode*: ACM, Washington, DC, USA, pp. 76-82.

RCFL (2007). *Annual report for fiscal year 2007*.

RCFL (2008). *Annual report for fiscal year 2008*.

Reed, C. (1990-91). 'As quoted in Casey E., 2004. Digital evidence and computer crime: forensic science, computers and the internet. 2nd ed. London: Academic press'.

Reed, C. & Angel, J. (2007). *Computer Law: The Law and Regulation of Information Technology*, Oxford University Press, Inc. , New York, NY, USA.

Reith, M., Carr, C. & Gunsch, G. (2002). 'An Examination of Digital Forensic Models', *International Journal of Digital Evidence*, vol. 1, no. 4, pp. 1-12.

Richard, G. & Roussev, V. (2006). 'Next-generation digital forensics', *Communications of the ACM*, vol. 49, no. 2, pp. 76-80.

Rogers, M. & Seigfried, K. (2004). 'The future of computer forensics: a needs analysis survey', *Computers & Security*, vol. 23, no. 1, pp. 12-16.

Royal, P., Halpin, M., Dagon, D., Edmonds, R. & Lee, W. (2006). 'PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware', *The 22nd Annual Computer Security Applications Conference*: IEEE Computer Society, Dec 26 Miami Beach, FL, USA pp. 289-300.

RSA (2011). 'The Current State of Cybercrime and What to Expect in 2011', *RSA 2011 cybercrime trends report*.

NTFSINFO (2006). 'NTFSINFO', version: 1.0. Retrieved from <http://technet.microsoft.com/en-us/sysinternals/bb897424>.

Strings (2009). 'Strings', version: 2.41. Retrieved from <http://technet.microsoft.com/en-us/sysinternals/bb897439>.

Santos, I., Penya, Y. K., Devesa, J. & Bringas, P. (2009). 'N-grams-based file signatures for malware detection', pp. 317–320.

Seewald, A. K. & Gansterer, W. N. (2010). 'On the detection and identification of botnets', *Computers & Security*, vol. 29, no. 1, pp. 45-58.

Shabtai, A., Moskovitch, R., Elovici, Y. & Glezer, C. (2009). 'Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey', *Information Security Technical Report*, vol. 14, no. 1, pp. 16-29.

Shafiq, M. Z., Khayam, S. A. & Farooq, M. (2008). 'Embedded Malware Detection Using Markov <i>n</i> -Grams', in D. Zamboni (ed.), *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer Berlin / Heidelberg, vol. 5137, pp. 88-107.

Shankarapani, M., Kancherla, K., Ramammoorthy, S., Movva, R. & Mukkamala, S. (2010). 'Kernel machines for malware classification and similarity analysis', *The 2010 International Joint Conference on Neural Networks*: IEEE Computer Society, Jul 18-23 Barcelona pp. 1-6.

Sharif, M., Yegneswaran, V., Saidi, H., Porras, P. & Lee, W. (2008). 'Eureka: A Framework for Enabling Static Malware Analysis', in S. Jajodia & J. Lopez (eds), *Computer Security - ESORICS 2008*, Springer Berlin / Heidelberg, vol. 5283, pp. 481-500.

Shetty, S. (2004). *Protocol-level malware scanner*, Google Patents, patent, 6772345, US.

Singh, N. (2007). 'Online Frauds in Banks with Phishing', *Journal of Internet Banking and Commerce*, vol. 12, no. 2, pp. 1-27

Skoudis, E. & Zeltser, L. (2003). *Malware: Fighting Malicious Code*, Prentice Hall PTR.

SPAMfighter News (2011). 'Alliance of ZeuS-SpyEye Resulting in the Publication of First Toolkit in the Underground Market'.

Stabek, A., Watters, P. & Layton, R. (2010). 'The Seven Scam Types: Mapping the Terrain of Cybercrime', *Cybercrime and Trustworthy Computing Workshop*: IEEE Computer Society, 19-20 July, Ballarat, VIC, pp. 41-51.

Stang, D. (2010). *Detection Errors and Scanner Performance*, retrieved 1 March 2011. Retrieved from <http://www.upublish.info/Article/Detection-Errors-and-Scanner-Performance/343804>.

Staniford, S., Moore, D., Paxson, V. & Weaver, N. (2004). 'The top speed of flash worms', *The ACM workshop on Rapid malcode* ACM, October 29-29, Washington DC, USA

Stolfo, S., Wang, K. & Li, W.-j. (2005). *Fileprint analysis for malware detection*, Columbia Univesrity.

Stolfo, S., Wang, K. & Li, W.-J. (2007). 'Towards stealthy malware detection', in *Malware Detection*, vol. 27, pp. 231-249.

Sun, L., Versteeg, S., Boztaş, S. & Yann, T. (2010). 'Pattern Recognition Techniques for the Classification of Malware Packers', in R. Steinfeld & P. Hawkes (eds), *Information Security and Privacy*, Springer Berlin / Heidelberg, vol. 6168, pp. 370-390.

Sung, A. H., Xu, J., Chavez, P. & Mukkamala, S. (2004). 'Static analyzer of vicious executables (SAVE)', *20th Annual Computer Security Applications Conference,* : IEEE Computer Society, Dec 6-10, Tucson, AZ, USA, pp. 326-334.

SWGDE (2000). 'Proposed Standards for the Exchange of Digital Evidence', *Forensic Science Communications, Digital Evidence:Standards and Principles*, vol. 2, no. 2.

Symantec Enterprise Security (1997). 'Understanding Heuristics: Symantec's Bloodhound Technolog', *Virus Bulletin*, vol. XXXIV.

Symantec Enterprise Security (2007). *Trojan.Vundo*, Symantec Enterprise Security,. Retrieved from.

Symantec Enterprise Security (2009a). 'Symantec Global Internet Security Threat Report Trends for 2008', *Symantec Enterprise Security*, vol. XIV.

Symantec Enterprise Security (2009b). *Trojan.Vundo.B*, Symantec Enterprise Security,. Retrieved from <http://www.symantec.com/security_response/writeup.jsp?docid=2005-042810-2611-99>.

Symantec Enterprise Security (2010). 'Symantec Internet Security Threat Report: Trends for 2009', *Symantec Enterprise Security*, vol. XV.

Symantec Enterprise Security (2011a). 'Symantec Internet Security Threat Report: Trends for 2010', *Symantec Enterprise Security*, vol. 16.

Symantec Enterprise Security (2011b). *Symantec Report on Attack Kits and Malicious Websites*, Symantec Enterprise Security White paper, Retrieved from.

Szor, P. (2005). *The Art of Computer Virus Research and Defense*, Addison-Wesley Professional.

Tamada, H., Okamoto, K., Nakamura, M., Monden, A. & Matsumoto, K.-i. (2006). 'Dynamic software birthmarks based on API calls', *IEICE Transactions on Information and Systems*, vol. 89, no. 8, pp. 1751-1763.

Tang, K., Zhou, M.-T. & Zuo, Z.-H. (2010). 'An Enhanced Automated Signature Generation Algorithm for Polymorphic Malware Detection', *Journal of Electronic Science and Technology*, vol. 8, no. 2, pp. 114-121.

Technology, B. A. (2006). *White paper*, BitDefender Antivirus Retrieved from.

Tian, R., Islam, R., Batten, L. & Versteeg, S. (2010). 'Classification of Malware Based on String and Function Feature Selection', *The International Conference on Malicious and Unwanted Software (MALWARE)*, : IEEE Computer Society, 19-20 July, Nancy, Lorraine, pp. 23-30.

Townsend, K. (2010). 'Anti-virus: a technology update', *Infosecurity*, vol. 7, no. 6, pp. 28-31.

TreadwellZhou, S. & Zhou, M. (2009). 'A heuristic approach for detection of obfuscated malware', *Intelligence and Security Informatics ISI09 IEEE International Conference on*: IEEE Computer Society, June 8-11, Richardson, TX, USA, pp. 291-299.

Trusteer (2009). *Measuring the in-the-wild effectiveness of Antivirus against Zeus*, New York, NY.

Turville, K., Yearwood, J. & Miller, C. (2010). 'Understanding Victims of Identity Theft: Preliminary Insights', *Cybercrime and Trustworthy Computing Workshop*: IEEE Computer Society, 19-20 July, Ballarat, VIC, pp. 60 - 68.

Vacca, J. (2005). *Computer forensics: computer crime scene investigation*, Networking Series, Charles River Media; 2 edition.

Varol, C. & Bayrak, C. (2011). 'Estimation of quality of service in spelling correction using Kullback-Leibler divergence', *Expert Systems with Applications*, vol. 38, no. 5, pp. 6307-6312.

Vassil, R. (2009). 'Hashing and Data Fingerprinting in Digital Forensics', vol. 7, pp. 49-55.

Vasudevan, A. & Yerraballi, R. (2006). 'Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation', *The 29th Australasian Computer Science Conference*: Australian Computer Society, Inc., Hobart, Australia, vol. 48, pp. 311-320.

Venkatraman, S. (2009). 'Autonomic Context-Dependent Architecture for Malware Detection', *e-Tech 2009, International Conference on e-Technology*: International Business Academics Consortium, 8-10 Jan, Singapore, pp. 2927-2947.

Venkatraman, S. (2010). 'Self-Learning Framework for Intrusion Detection', *The International Congress on Computer Applications and Computational Science*: 4-6 Dec, Singapore, pp. 517-520.

Venkatraman, S. (2011). 'A Framework for ICT Security Policy Management', in E. Adomi (ed.), *Frameworks for ICT Policy: Government, Social and Legal Issues*, IGI Global Publishers, Hershey, New York, pp. 1- 14.

VX Heavens (2011). *VX Heavens Site*, retrieved 2/3 2011. Retrieved from <http://vx.netlux.org/>.

Wang, C., Pang, J., Zhao, R., Fu, W. & Liu, X. (2009). 'Malware Detection Based on Suspicious Behavior Identification', *First International Workshop on Education Technology and Computer Science*: IEEE Computer Society, Mar 07-08, Wuhan, Hubei, China, vol. 2, pp. 198-202.

Wang, C., Pang, J., Zhao, R. & Liu, X. (2009). 'Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior', *The International Conference on Communication Software and Networks*: IEEE Computer Society, 27-28 Feb, Macau, China pp. 544-548.

Wang, H., Bell, D. & Murtagh, F. (1999). 'Axiomatic Approach to Feature Subset Selection Based on Relevance', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 3, pp. 271-277.

Wang, X. G., Tang, Z., Tamura, H., Ishii, M. & Sun, W. D. (2004). 'An improved backpropagation algorithm to avoid the local minima problem', *Neurocomputing*, vol. 56, pp. 455-460.

Watters, P. (2009). 'University incorporated: implications for professional information security education', *Corporate Governance*, vol. 9, no. 5, pp. 564 - 572.

Watters, P. & McCombie, S. (2011). 'A methodology for analyzing the credential marketplace', *Journal of Money Laundering Control*, vol. 14, no. 1, pp. 32 - 43.

Wing Wong (2006). 'Analysis and detection of metamorphic computer viruses', Master Degree thesis, San Jose State University.

Data mining: Practical machine learning tools and techniques (2010). 'Data mining: Practical machine learning tools and techniques', version: 3.6.4. Retrieved from <http://www.cs.waikato.ac.nz/ml/weka/>.

Wolf, L. & Shashua, A. (2005). 'Feature Selection for Unsupervised and Supervised Inference: The Emergence of Sparsity in a Weight-Based Approach', *J. Mach. Learn. Res.*, vol. 6, pp. 1855-1887.

WinHex Hex Editor (2010). 'WinHex Hex Editor', version: 12.8. Retrieved from.

Xiao, J., Liu, B. & Wang, X. (2007). 'Exploiting Word Positional Information in Ngram Model for Chinese Text Input Method', *Journal of Information and Computing Science*, vol. 2, no. 3, pp. 215-222.

Xu, J., Sung, A., Chavez, P. & Mukkamala, S. (2004). 'Polymorphic malicious executable scanner by API sequence analysis', *The Fourth International Conference on Hybrid Intelligent Systems*: 5-8 Dec, vol. 378-383.

Yanfang, Y., Tao, L., Qingshan, J. & Youyu, W. (2010). 'CIMDS: Adapting Postprocessing Techniques of Associative Classification for Malware Detection', *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 3, pp. 298-307.

Yang, L., Karim, R., Ganapathy, V. & Smith, R. (2010). 'Improving NFA-Based Signature Matching Using Ordered Binary Decision Diagrams Recent Advances in Intrusion Detection', in S. Jha, R. Sommer & C. Kreibich (eds), Springer Berlin / Heidelberg, vol. 6307, pp. 58-78.

Yao, D. & Liu, X. (2011). 'Research on the Cyber Terrorist Attacks and its Impacts on Information Infrastructure Security', *Advanced Materials Research, Computational Materials Science*, vol. 268 - 270, pp. 2108-2115.

Ye, Y., Wang, D., Li, T. & Ye, D. (2007). 'IMDS: intelligent malware detection system', *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*: ACM, San Jose, California, USA, pp. 1043-1047.

You, I. & Yim, K. (2010). 'Malware Obfuscation Techniques: A Brief Survey', *International Conference on Broadband, Wireless Computing, Communication and Applications*: IEEE Computer Society, pp. 297-300.

Zhang, F., Qi, D. & Hu, J. (2010). 'Using IRP for malware detection', *The 13th international conference on Recent advances in intrusion detection*: Springer-Verlag, Ottawa, Ontario, Canada, pp. 514-515.

Zhu, Z., Ong, Y.-S. & Dash, M. (2007). 'Wrapper-Filter Feature Selection Algorithm Using a Memetic Framework', *IEEE transactions on systems man and cybernetics Part B Cybernetics*, vol. 37, no. 1, pp. 70-76.

Zitouni, I. (2007). 'Backoff hierarchical class n-gram language models: effectiveness to model unseen events in speech recognition', *Computer Speech & Language*, vol. 21, no. 1, pp. 88-104.