ANALYSIS AND IMPROVEMENTS OF BEHAVIOUR-BASED MALWARE DETECTION MECHANISMS

by

NADA ALRUHAILY

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
September 2017

ABSTRACT

The massive growth of computer usage has led to an increase in the related security concerns. Malware, such as Viruses, Worms, and Trojans, have become a major issue due to the serious damages they cause. Since the first malware emerged, there has been a continuous battle between security researchers and malware writers, where the latter are constantly trying to evade detection by adopting new functionalities and malicious techniques.

This thesis focuses on addressing some of the concerns and challenges encountered when detecting malware, based on their behavioural features observed; for each identified challenge, an approach that addresses the problem is proposed and evaluated. Firstly, the thesis provides an in-depth analysis of the underlying causes of malware misclassification when using machine learning-based malware detectors. Such causes need to be determined, so that the right mitigation can be adopted. The analysis shows that the misclassification is mostly due to changes in several malware variants without the family membership or the year of discovery being a factor. In addition, the thesis proposes a probabilistic approach for optimising the scanning performance of Forensic Virtual Machines (FVMs); which are cloud-based lightweight scanners that perform distributed monitoring of the cloud's Virtual Machines (VMs). Finally, a market-inspired prioritisation approach is proposed to balance the trade-off between the consumption of VMs' resources and accuracy when detecting malware on the cloud's VMs using Virtual Machine Introspection-based lightweight monitoring approaches (e.g. FVMs). The thesis concludes by highlighting future work and new directions that have emerged from the work presented.

This work is dedicated to my beloved ones:

To my dearest parents, Massoud and Fatimah, for their unconditional love and sacrifices;
To my treasured husband, Abdullah for his support, and my precious children, Leen and Mohammed;
To my dear brothers, Waleed and Abdullah;
To my dear sisters, Shatha, Banan and Haneen.

ACKNOWLEDGEMENTS

First and foremost, I thank *Allah* for giving me the strength, patience and determination to reach this precious moment, in which I find myself adding the final touches to my PhD thesis, an indication of a successful completion of a remarkable journey. My PhD in the School of Computer Science at the University of Birmingham was a journey full of exciting achievements; difficult, yet, special moments. I was lucky enough to meet so many great people who will always be a part of such an exceptional experience. The first person that I wish to express my sincere gratitude to, is my supervisor *Dr. Tom Chothia*, who helped me to become a confident researcher, starting from developing my ideas, writing and structuring my own papers through to presenting the work at several of the best conferences. Tom, I cannot thank you enough for all your guidance during my Master's degree and PhD, for your invaluable ideas, and most importantly, for your infinite patience.

I would like also to express my special appreciation to *Dr. Behzad Bordbar*, who was my second supervisor until September 2016. Thank you for the encouraging discussion from the start of my PhD till the day you left. Special thanks also goes to my Thesis Group meeting members for all their suggestions and valuable insights: *Dr. Peter Tino*, *Dr. Rami Bahasoon* and *Dr. Marco Cova* (who was one of the group members until September 2016).

Finally, my deepest and unlimited gratitude goes to my family for their unconditional love and support throughout my whole life; this PhD research would have been impossible without them. I am indebted to my parents, *Massoud* and *Fatimah*, who have been tremendously supportive in every aspects of my life; to my husband, *Abdullah*, who has

the greatest faith in me and has provided undaunted support; to my brothers, sisters and friends in the school of computer science for their persistent encouragement, and most importantly, to my children, *Leen* and *Mohammed*; without their beautiful smiles, pure souls, and innocent giggles, my PhD journey would have been monochrome.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Malware Malicious Software

MBR Master Boot Record

CARO Computer Anti-virus Researchers Organisation

ML Machine Learning

VMI Virtual Machine Introspection

FVMs Forensic Virtual Machines

AI Artificial Intelligence

SVM Support Vector Machine

DT Decision trees

TP True positive

TN True negative

DT False positive

FN False negative

G-mean Geometric mean

ROC curve receiver operating characteristic curve

$\mathbf{AUC_{ROC}}$ Area under the ROC curve

DLL Dynamically Linked Libraries

VMM Virtual Machine Monitor

NIST National Institute of Standards and Technology

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastructure as a Service

VENOM Virtualised Environment Neglected Operations Manipulation

AV Anti-Virus

IDS Intrusion Detection System

MI Mutual Information

API Application Programming Interface

Opcodes Operation Codes

Bagging Bootstrap aggregating

EBBag Exactly Balanced Bagging

CSP Cloud Service Provider

CHAPTER 1

INTRODUCTION

1.1 Overview

Computers and human bodies are similar in the sense that both of them could be affected by various illnesses which can be due to a virus infection. Computer and biological viruses are both considered to be self-replicating entities, which use a host as a means to spread and replicate; and as a result of their actions, various kinds of damage could be incurred to the host. Both are also similar in the sense that they could adopt a number of techniques that make the process of identifying, eliminating and defeating them a difficult task [73].

However, computer viruses are only one form of many '*malicious software*' that can be produced with the purpose of stealing, manipulating, or erasing users' data and other malicious activities. In fact, with the massive growth of computer usage, the number of different computer-based attacks accordingly grows, causing considerable damage, and global data loss incidents. For instance, on the 16$^{\text{th}}$ August 2012, a malicious software named *Shamoon* [170], known also as *Disttrack*, targeted the Saudi Arabian oil company *'Aramco'* [145]. Once a machine is infected, the malicious software will overwrite the *Master Boot Record* (MBR) causing severe damage, where the infected machine becomes unbootable. Another version (a.k.a variant) of the malicious software had also been seen on the 23$^{\text{th}}$ November 2016, with similar, but improved functionalities [171].

Moreover, in June 2010, a malicious software known as *Stuxnet* revealed a new genera-

tion of targeted attack, or a campaign; its purpose was not data erasure or financial gain, which is the usual aim of most similar attacks; rather, the main aim was to destroy a specific military target, represented in the Iranian nuclear centrifuges. It has been recorded as the first cyberwarfare weapon [96].

Damage resulting from malicious software is not limited to computers only; it may extend to a considerable real life impact, even to the life of individuals. An example is the US government's Office of Personnel Management data breach of 2015, which resulted in the leakage of 21.5 million records of highly sensitive personal information, including fingerprints [168]; such sensitive data can be used for numerous malicious purposes such as identity theft.

It was possible to perform all of the aforementioned attacks by using 'malicious software', or for short a *Malware*, which facilitate the attack and the planned malicious goals.

1.1.1 Malicious Software (Malware)

The term **malware** is short for **mal**icious soft**ware**; it includes any software that enters a system to perform activities without the user's consent and with a malicious intention. The term covers a wide range of classes and types; the most common examples are:

- **Virus**

  This type of malicious software needs a carrier (a.k.a. host program) to spread to other machines. When the host application is executed the virus starts its malicious activity by infecting other files, modifying and damaging the data on the infected machine.

- **Spyware**

  This program is installed without the user's permission; it gathers information related to the user's activity, such as browsing information, without the user's knowledge, to be sent to a remote server or user. Spyware can also download additional malicious applications from the internet.

- **Trojans**

  These are misleading programs that look like useful and legitimate programs; however, they are actually made with a malicious intention. They can be used to open a backdoor on the infected machine, allowing an unauthorised access by a malicious user or a program; this results in the data on the infected machine being damaged or stolen.

- **Worms**

  A worm is a standalone, self-replicating program that does not need to attach itself to a host application in order to propagate. Such programs use a computer network to cause major damage and slow down traffic within the infected network; their typical payload is to install a backdoor, which can be used by the attacker to control and gain continued access to the infected machine.

- **Ransomware:**

  When a machine is infected with such malicious software, the data become inaccessible, either because the system has been locked, or the data stored has been encrypted until a ransom is paid to the attacker.

Other malware classes also exist, such as downloaders, wipers and misleading applications (a more extensive introduction can be found in [173]). This section has listed a number of malware classes; however, it is important to note that some of the above classes are not mutually exclusive, which implies that the features of a number of different classes could be observed in a single malware [62].

Malware Naming

Although anti-malware vendors may have different names for the same sample, they usually share a similar naming convention where malware can be grouped into families based on some similarities (code similarity as an example); in turn, each malware family can have a number of variants, where each variant represents a new strain that is slightly

modified. This naming convention was originally based on the malware naming scheme formed in 1991 by the Computer Anti-virus Researchers Organisation (CARO) [30].

Figure 1.1 shows an example of the naming scheme followed by the company Symantec; it consists of a prefix, the family name, and a suffix [162]. Throughout this thesis, *Symantec's Listing of Threats and Risks* [169] has been used as a source of malware discovery date, type, malware family and variant name.

Figure 1.1: Symantec malware naming scheme.

1.1.2 Malware Detection

According to the latest *Symantec's Security Response Publications* [172], the number of malware variants currently released can be as high as 96.1 million variants in one month (as shown in Figure 1.2). In order to protect systems from malware infections, the use of a malware detector is essential. Malware detectors are programs which scan files and identify whether or not the files are malicious. Although malware detectors can be based on different techniques, generally, they fall into two main categories: *signature-based* and *behaviour-based* malware detectors [82, 117]. In the signature-based malware detectors, a hash, a byte sequence, or more complex signatures in the inspected file is matched to a database of signatures of known malicious files; the database is maintained and updated regularly. Those malware detectors, which are signature based, usually have an extremely low false positive rate [67]. However, this technique of detection cannot protect from new malware threats (a.k.a zero day malware), as they have not been inspected, therefore, their signature has not yet been added to the database. Furthermore, signature-based detectors are known for their considerable usage of resources during scaning [150]; limiting their practical usage on lightweight computer devices with limited amount of resources.

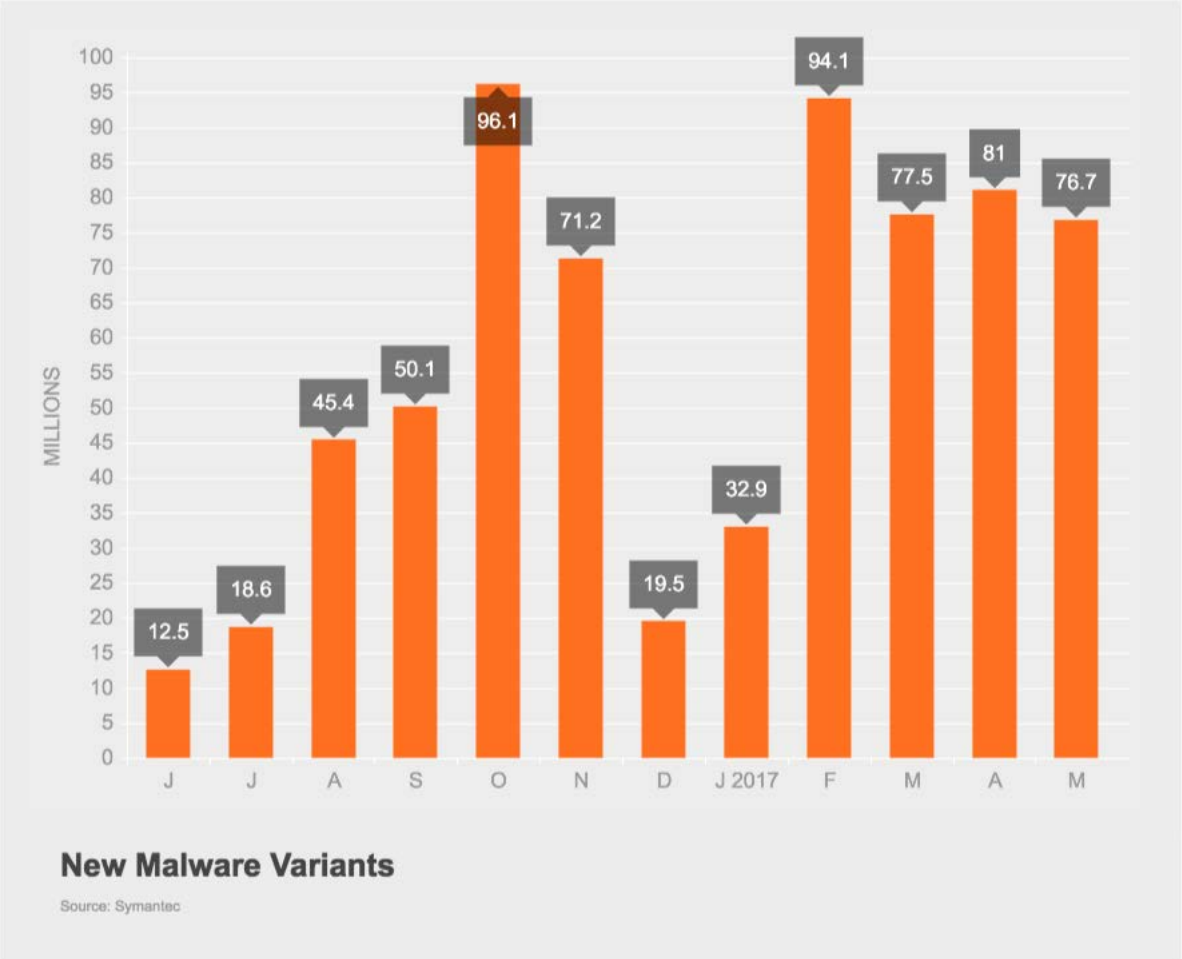


Figure 1.2: New Malware variants discovered monthly, adopted from [172].

As for behaviour-based malware detectors, instead of identifying a file as malicious based on its signature, the inspected file will be identified as malicious if it performs any malicious behaviour. Techniques based on behavioural detection can generate behavioural models of malware that, in turn, are used to identify previously unseen malware samples by using advanced methods and algorithms such as *machine learning* (ML). ML-based detection systems show that they could provide a high detection rate when recognising non-previously seen malware samples [65, 209, 72, 97]. However, some malware can go undetected as a result of changes in behaviour compared to the previously seen data. **Although a large amount of work has been conducted by malware researchers to propose new ML-based detection systems (As shown later on in**

Chapter 2 p. 26), the reasons behind the misclassification of malware (a.k.a. non-detection of malware samples) were rarely documented, resulting in an ambiguity when trying to interpret the resulting misclassification rate.

Malware infection is not limited to classical computing only; the wide use of cloud computing and its services also makes it a potential target. Malware detectors that are placed inside the monitored system, and implement one of the known malware detection techniques (e.g. signature matching, or ML techniques), have their clear limitations in terms of resources, and overhead costs when used to detect malware on the cloud. This is mainly due to the fact that such detectors could cause a disk and network bandwidth saturation on the hosting system, leading to a situation that is referred to, by the leading security vendors, as 'AV storm' [174, 78]. As a result, a number of cloud-based malware detection techniques have been proposed to mitigate the costly use of resources. One of the recent techniques which has been used actively to propose lightweight behaviour-based malware detection mechanisms is *Virtual Machine Introspection* (VMI)) [63]. It relies on virtualisation technology that allows division of the resources between multiple instances of virtual machines (VMs). VMI techniques enable the software that hosts the cloud VMs (or a guest VM) to monitor the internal state of another VM from the outside, when having sufficient privilege. The external monitoring of VMs adds an extra level of security (i.e. the isolation offered between the malware detector and the inspected VM). Examples of such lightweight detection mechanisms that are based on VMI technology include, Bitdefender Hypervisor Introspection (HVI) [22], and the lightweight technique used in Cloudidea [57].

The task of identifying the malicious behaviour on the cloud using VMI can also be subdivided among small scanners or mini-VMs (referred to as *Forensic Virtual Machines* (FVMs)), which perform distributed monitoring to reduce further the resources used during the scan. Each FVM is responsible for identifying a single symptom; examples of symptoms can be turning off a firewall, adding a process to the system auto run list, or rebooting a machine. The movement of FVMs from one VM to another is determined by

a distributed algorithm, called a *mobility algorithm*. Mobility algorithms can vary from simple to dynamic algorithms, such as scanning according to a random movement, a predetermined order, or more advanced scanning algorithms. Each FVM is initialised with a specific mobility algorithm; **however, to get the full potential of FVMs, there is a need to identify the most beneficial mobility algorithm which will promote efficient scanning and detection of malware infections**.

Generally speaking, techniques that are based on VMI (e.g. FVMs) offer a lightweight monitoring option for the cloud's VMs; they help in minimising the consumption of the VMs' resources and in reducing the impact on the VMs performance. **However, as such techniques depend on detecting behavioural symptoms that can be seen on both infected and non-infected systems, sometimes a clear-cut distinction would not be easy to identify, leading to wrongly flagging a number of VMs as infected.**

In terms of the features used in the detection process, they can be extracted using either, static or dynamic analysis techniques. In static analysis, the focus is on the disassembled code, where the features and patterns used in the detection process are extracted without running the malicious code. This technique provides a high code coverage, where all the execution paths could be examined. However, malware writers commonly use custom packers to protect their internal code from being examined [201, 180]; they may also develop malicious binaries in a way that makes parsing the instructions and understanding the binary control and data flow an extremely complex task. In dynamic analysis, on the other hand, samples are examined during the execution; thus, this method is not vulnerable to such aforementioned malware obfuscation techniques. Throughout this thesis dynamic analysis techniques were used to generate malware features.

1.2 Research Questions and Objectives

The challenges and concerns surrounding the detection of malware, as mentioned in the previous section, motivate the work presented in this thesis. In general, this thesis is concerned with the detection of malware, based on their exhibited behaviour and symptoms. It develops a better understanding of misclassification causes of malware when using ML-based malware detection techniques, by investigating possible misclassification patterns. It also addresses some challenges in using VMI-based techniques to detect malware according to their observed symptoms. In particular, this thesis proposes solutions that answer the following research questions:

(RQ1): What are the reasons behind the misclassification of malware?
Machine learning-based malware detection systems have been suggested as a replacement for signature-based detection methods. These systems have shown that they can provide a high detection rate, even when recognising previously unseen malware samples. However, in those systems that are based on behavioural features, some new malware can go undetected, and therefore, is classified wrongly as benign. The misclassification that occurs might be due to the effect of the choice of the machine learning method, or due to noisy data, or due to changes in the underlying features of malware, compared to the training data, where re-training of the system is required. Determining the misclassification causes helps in interpreting the resulting misclassification rate, and therefore, identifying the appropriate mitigation method. Thus, the objective here is to identify the underlying causes of malware misclassification by answering the following sub-questions: (i) *Does misclassification increase over a period of time?* (ii) *Do changes that affect classification occur in malware at the level of families, where all instances that belong to certain families are hard to detect?* (iii) *Alternatively, can such changes be traced back to certain malware variants instead of families?* (iv) *When misclassification occurs can we find the reason for it?*

(RQ2): How can malware scanning performance of Forensic Virtual Machines be optimised ?

Forensic Virtual Machines is an architecture for using Virtual Machine Introspection technology. It benefits from mini-VMs that perform distributed monitoring to detect symptoms of malicious behaviour from outside the target VM. Scanning using these mini-VMs consumes fewer resources than a full malware scan would, as each FVM examines a single symptom. In order to reduce the creation cost of FVMs, each is shared among a number of VMs; and one of many distributed algorithms (referred to as mobility algorithms) is embedded in each FVM to specify how an FVM chooses its next scanning target. The objective here is *to provide a way to analyse those algorithms in order to nominate the most beneficial one. Although multiple algorithms have been introduced in the literature, there is no work that addresses the issue of identifying the most beneficial algorithm prior to the FVM creation process, which is considered a resource-intensive task.*

(RQ3): How can VMI-based lightweight monitoring approaches (e.g. FVMs) be integrated in a cloud-based malware detection system to develop efficient and effective detection of malware infections?

Traditional Anti-Virus solutions are known for their inability to identify non-previously seen malware infections that have not been inspected yet; they are also known for their considerable usage of resources limiting their usefulness especially on the cloud. In contrast, Virtual Machine Introspection-based lightweight malware monitoring approaches (e.g. FVMs), which monitor VMs' state periodically in order to detect any malicious activities, are known for their minimal impact on the cloud's VMs. They consume fewer VM's resources during the scan than a full malware scan would normally require. This is mainly because they carry out an external inspection of VMs, and they examine malicious behaviour that can be shared among multiple malware families as opposed to matching signatures. However, the fact that such techniques are based on identifying symptoms that exist in both malicious and normal behaviour, but

in different proportions or combinations, makes them prone to false alarms, thereby limiting their effectiveness. Therefore, the objective here is *to address the problem of balancing the trade-off between the consumption of VMs' resources, and the detection accuracy (in terms of false alarms generated), when using such VMI-based lightweight monitoring approaches to detect malware infections in the cloud environment.*

The research questions, and objectives above delimit the scope of the research work and describe the primary aims of this thesis, which focuses on providing a better understanding of malware misclassification and introducing new solutions to increase the efficiency and effectiveness of malware detection in the cloud.

1.3 Research Contributions

The work described in this thesis contributes to the field of malware detection by providing an in-depth analysis of the causes of malware misclassification, as well as introducing novel solutions that increase the efficiency and effectiveness of malware detection in the cloud using VMI-based malware detection techniques.

In summary, this thesis makes the following contributions:

1. Developing a better understanding of the misclassification rate of malware detection systems that are based on ML techniques, by using a typical implementation of a machine learning-based detection system. The system is used to analyse misclassified malware instances, and investigate whether there were recognisable patterns across these misclassifications. The results of this analysis can help researchers to interpret other papers that present the detection rate of an ML-based system as their main result. They can also shed some light on how these detection systems will perform over time, and why some malware avoid detection.

2. Developing a probabilistic approach that helps in identifying the most beneficial mobility algorithm prior to the initialisation of FVMs. The evaluation conducted

showed that there was a considerable difference in the scanning performance when using different mobility algorithms; consequently, it was possible to identify the most beneficial algorithm, given parameters that represent the scanned cloud environment (e.g. the number of VMs, the deployed FVMs and the cost of false positives and false negatives). Checking which algorithm is the best prior to the initialisation of FVMs can help in identifying the most critical set of VMs with the minimum use of the scanning resources, thereby resulting in an efficient use of such a critical system.

3. Proposing a market-based prioritisation approach that utilises lightweight VMI-based malware scanners to perform an efficient and effective detection of malware infections. The proposed approach consists of two layers of protection for a more in-depth scan of the cloud's VMs. The main novelty of this technique is that it utilises a market mechanism that guides the lightweight scanners to prioritise the AV scanning process, by deciding which VM should be thoroughly scanned and when. It will then trigger a full malware scan on a pre-defined percentage of the most critical VMs. The evaluation conducted shows that the approach provides a cost-effective scanning method, while being able to confirm the infection status of the most critical set of VMs, thus balancing the trade-off between the consumption of VMs resources, and the false alarms generated.

1.4 Publications Based on the Thesis

- Alruhaily N., Bordbar B. and Chothia T. (2015). 'Analysis of Mobility Algorithms for Forensic Virtual Machine Based Malware Detection'. In Trustcom/BigDataSE/ISPA, 2015 IEEE, pages 766-773 [6], (acceptance rate: 28.4%). This contribution is presented in Chapter 5.

- Alruhaily N., Bordbar B. and Chothia T. (2017). 'Towards an Understanding of the Misclassification Rates of Machine Learning-based Malware Detection Systems'.

In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017), pages 101-112 [7], (acceptance rate: 21%). This contribution is presented in Chapter 4.

- Alruhaily, N., Mera-Gómez, C., Chothia, T., and Bahsoon, R. (2017). 'A Market-Based Approach for Detecting Malware in the Cloud via Introspection'. In the 15th International Conference on Service-Oriented Computing (ICSOC 2017), pages 722-730 [8], (acceptance rate: 18%). This contribution is presented in Chapter 6.

1.5 Thesis Structure

An overview of the remainder of this thesis is outlined below.

Chapter 2

This chapter introduces necessary definitions and discusses the background knowledge required to follow the research, and the results obtained in the subsequent chapters of this thesis. It also discusses the related state-of-the-art work on the areas of malware detection, and provide a comprehensive literature review. However, works that are directly related to each of the proposed contributions are explored and discussed amply in each associated chapter.

Chapter 3

In this chapter, we discuss our approach towards developing a representative malware corpus which was used in the subsequent chapters of this thesis, in addition to describing how the information collected could be transformed into an intermediate representation that is ready to be used for detecting malware. The chapter concludes by giving some statistics related to the knowledge-base that has been generated.

Chapter 4

This chapter presents an in-depth analysis of causes of malware misclassification, by conducting multiple experiments, which investigate whether there were recognisable

patterns across the misclassified malware instances. This chapter shows that the changes in malware behaviour, which lead to malware instances being misclassified, are mostly due to changes in several malware variants without the family membership or the year of discovery being a factor.

Chapter 5

In this chapter, we discuss the development of a probabilistic approach that is proposed to optimise the Forensic Virtual Machines monitoring process. The approach helps in nominating the most beneficial mobility algorithm based on probability theory and Bayes' theorem, and by utilising the malware knowledge-base developed in Chapter 3. The approach was evaluated through a comparison of the scanning cost resulting from using a number of mobility algorithms, which range from simple to more advanced dynamic ones.

Chapter 6

This chapter discusses our solution towards addressing the trade-off problem between detection accuracy, and consumption of VMs' resources when detecting malware in the cloud, using VMI-based light-weight monitoring techniques. The chapter introduces a market-based prioritisation approach, which utilises two layers of protection (i.e. lightweight VMI-based technique and full malware scanning) to promote an efficient and effective scanning of malware infections in the cloud.

Chapter 7

This chapter elaborates on how the solutions and the approaches proposed have answered the research questions, and delivered contributions to the field of knowledge. We also present the potential future work and research directions that have emerged from this thesis. The thesis then concludes by presenting a brief summary.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

In general, two phases distinguish the malware detection process: the feature generation and extraction phase, which can be accomplished by analysing malware samples using commonly known malware analysis techniques, and the actual detection phase. In this chapter, we firstly present an introduction related to those two phases: malware analysis including their different techniques, and malware detection. We then present some techniques which have been widely used to implement malware detectors; in particular, we introduce the following: machine learning, and virtual machine introspection that is enabled by virtualisation technology in cloud computing. The chapter then presents some preliminary definitions regarding probability theory, information theory and market mechanisms that are required to follow the work and contributions presented in the subsequent chapters of this thesis. We should highlight that the work that is directly related to the contributions made in this thesis, are discussed amply in the main contribution chapters (i.e. Chapters 4, 5 and 6).

2.1 Malware Analysis Techniques

Many studies have been conducted since malware first emerged, and these have introduced several analysis tools and techniques to assist researchers to attain a better understanding of malware, and therefore to enable them to implement efficient detection systems that can

minimise the dangers of malware, or even prevent the infection. In general, such analysis techniques fall into three main categories: *static analysis, dynamic analysis* and *hybrid analysis techniques* (combining both static and dynamic). In static analysis, the focus is on disassembled code, in which the features and the pattern used in the detection process are extracted without running the malicious code; then Control Flow Graphs (CFG) and Data Flow Graphs (DFG) are constructed based on the disassembled code. Such graphs offer intermediate representation of the behaviour of the analysed binary. Various static features can also be extracted from the disassembled binary; such as imported API functions, binary strings and signatures, Opcodes (operation codes), among others [37]; a number of researchers have used these features as a basis for their detection systems (e.g. [4, 189]).

In dynamic analysis, the malicious binary is analysed by executing the code in an isolated environment [50, 15, 194] wherein various features can be extracted, including API calls, system traces and network traffic. Such features have been used to develop behaviour-based malware detection systems [56, 5, 68].

Both analytical techniques have advantages and disadvantages, and a significant quantity of diverse information about malware behaviour and actions can be obtained using either. One of the main advantages obtained when using a static analysis technique is that it provides a high degree of code coverage, which is not limited to one execution path, as in dynamic analysis. Conversely, static analysis is vulnerable to obfuscation techniques, wherein the malware writer might develop malicious binaries in a way that makes parsing instructions and understanding a binary control and data flow an extremely complex task. In addition, if the executable file is found to be packed with an unknown packer, the analyser might be rendered unable to complete the required analysis. Likewise, when using static analysis to analyse the malware, the analyser has to assess the entire code, which could incur considerable time and effort [15, 50].

In contrast, dynamic analysis can save analysers' time and effort, and facilitate the analysis process of the packed executables, without the need to determine the packer, since

the malware is analysed during the execution. However, several possible disadvantages could result from using such a technique. For example, malware might determine that it is running in a virtual environment, as such an environment is used for dynamic analysis to minimise the risk of infection; therefore, malware can alter its behaviour accordingly to be non-malicious. In addition, in most dynamic analyses, only a single execution path is examined, and although some studies have shown that this drawback can be overcome using a new approach to explore multiple execution paths [114], this method has not been broadly applied [15, 50].

As static analysis could be easily circumvented, and thus completely bypassed by using a number of obfuscation techniques (e.g. using a custom or an unknown packer), dynamic analysis has been widely used to develop more effective malware detection [88]. Therefore, in this thesis the focus will be on the dynamic analysis technique only. The next section gives an overview of the tools currently used for this type of analysis.

2.1.1 Dynamic Analysis Tools and Agents

As a result of the growing interest in understanding and analysing malware behaviour over the last decade, a number of tools were introduced to automatically analyse malicious binaries during execution. However, the quantity of information gathered from using such dynamic analysis tools can vary, depending on the approach applied. One of the most popular approaches is to monitor the function calls of a running system by implementing function-hooking techniques. These techniques intercept running processes to monitor the execution of the binary. In addition, other approaches are used in order to perform dynamic analysis; examples include information flow tracking, function parameter analysis and instruction trace [50].

Generally, dynamic analysis tools are based on one or more of these approaches: CWSandbox [194], Anubis [15, 16], Norman [124], ThreatExpert [175] and Cuckoo Sandbox [43], which are all examples of tools utilising one or more of these approaches to perform a dynamic analysis. Such tools (specially the sandboxes mentioned) share the

same functionality: executing malware in a controlled environment and monitoring the malware execution mainly by applying the *API hooking* technique in order to report the behaviour of a binary during the execution. The reports generated can be used in the development of different defence mechanisms [50].

Throughout this thesis we have utilised two widely used malware analysis tools in malware research community, namely: Anubis [14, 17, 29, 89], and Cuckoo Sandbox [137, 60, 68], to analyse malware samples dynamically during the executions phase and collecting their behaviour traces; therefore, further analysis or evaluation of the proposed methods could be conducted. Chapter 3 describes the process of collecting and analysing malware using these tools and gives an overview of the resulting information.

2.2 Malware Detection

In order to protect machines from a malware infection, the use of a malware detector (a.k.a malware scanner) is essential. Malware detectors are programs that scan files and identify whether they are malicious or not, based on some malicious activities or characteristics observed. Although malware detectors can utilise different techniques, generally speaking, they can fall into two main categories: *signature-based* and *behaviour-based* malware detectors [82, 117]. In the signature-based type, a hash, a byte sequence in the inspected file, or more advanced signature will be matched to a database that contains signatures of known malicious files. This database is maintained and updated regularly. Signature-based malware detectors commonly have a very low false positive rate; however, this detection technique cannot protect from new malware threats (a.k.a zero day malware), as they have not been inspected; thus their signature has not yet been added to the database [36]. Listing 2.1 shows two different types of traditional signatures: MD5 hashing and Byte sequence with wildcards '?' and disjunctions ($8a|86$), which are used to identify specific virus strains; namely `W32.Sality.AE` [164], `Paulus.1804` [182] and `W32.Virut.si` [198] respectively:

Listing 2.1: Different type of malware signatures

```
- W32.Sality.AE:
da2b0b17905e8afae0eaca35e831be9e

- Paulus.1804:
B9 D5 00 8B DE ?? ?? 27 06 53 ?? ?? 07 86 CA ?? ?? 86 CA 2E 88 07 4A ??

- ClamAV signature for W32.Virut.si
(8a|86) 06 66 (29|31) (c8|d0|d8|e8|f8) (86|88) 06 46
```

On behaviour-based detection systems, instead of identifying a file as malicious based on its static signature, the inspected file will be identified as malicious according to whether it exhibits any malicious behaviour [117]. Such systems generate rules or behavioural models of malware, for use in the detection process, to identify previously unknown malware samples. Those rules or models can be developed either manually by an expert [159], or automatically using advanced techniques, such as Artificial Intelligence (AI), machine learning (ML) algorithms, and data mining techniques [148]. This thesis does not attempt to provide an overview of all behaviour-based malware detection systems (see [82] for a comprehensive review); however, two behaviour-based detection approaches, along with the techniques that they have been based on, are covered in this chapter as they are the focus of this thesis: machine learning-based, and Virtual Machine Introspection-based malware detection approaches. These approaches, along with the techniques that they have been based on, are fully described in the following two sections, Section 2.3 and Section 2.4, respectively.

2.3 Machine Learning (ML)

In this section, we firstly provide some brief, but essential background on machine learning. The section starts by providing a definition and an overview of the technique, with a specific focus on the supervised learning and classification methods used to detect malware. Afterwards, the section presents several measures that are usually used to assess the performance of the outcomes of a machine learning method. The section is then finished

off by discussing how such a technique has been used to introduce more efficient malware detectors; it also discusses briefly the nature of malware evolution, and the effect of that on such detection systems.

According to Mitchell [113], the field of machine learning can be defined as the development and study of the algorithms used to enhance the process of computer programs' learning, based on some prior experience. The learning task involved could be referred to as *supervised*, when data labels are given during the learning process; if the labels cannot be made available during the learning phase, the learning task is then said to be *unsupervised*. Problems that can be solved using supervised learning algorithms include *classification* and *regression*, whereas unsupervised learning algorithms are generally used to solve problems such as *clustering*. The process of detecting malware can be considered as a supervised learning task (a classification in particular), as the aim is to predict the class of an 'unseen' sample after the system was fed, during the learning phase, with previously seen and 'labelled' malware or benign samples. Therefore, in the following sections we are focusing on classification tasks and methods only.

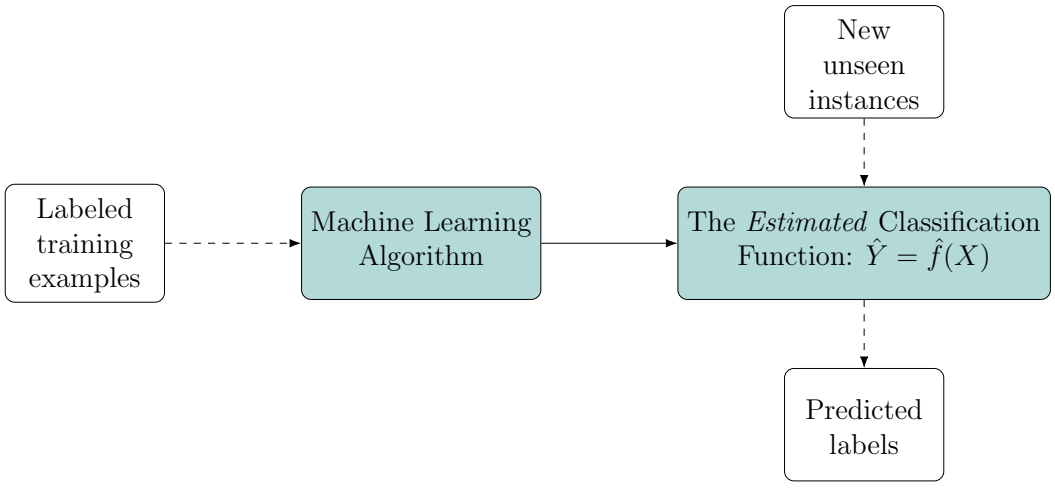


Figure 2.1: A classical classification problem.

2.3.1 Classification Methods

In classification tasks, the learning phase is commonly referred to as the *training phase*, while the process of classifying the new data is known as the *prediction, or testing phase*. Classification algorithms in the training phase construct a *classifier* which is a hypothesis h that gives an approximation of the true function f. Given an input space X, and a label space Y, the true function f is used to map an unlabelled input $x_n \in X$ to a predicted label (a.k.a class) $y_m \in Y$, such that $y_m = f(x_n)$. The prediction phase is concerned with identifying the newly unseen malware, based on the pattern and information extracted from the training phase. The meaningful pattern is usually extracted from labelled individuals (known as *instances*), which include malware and benign binaries in the case of the current scenario. Figure 2.1 demonstrates a classical classification task which aims to identify the new unlabelled data [118].

In the classification task, each instance belongs to a specific class and it is characterised by a number of features. Malware detection is usually considered a text classification problem, where each input x_i represents a vector of features, such that: $x_i = ({x_i}^1, {x_i}^2, {x_i}^3, \cdots, {x_i}^d)$, where d is the dimension of the data point x_i.

A number of ML algorithms are widely used for text classification purposes, such as Naive Bayes, Support vector machines [71], Decision trees [100], and Bagging algorithms [27]. Below, a brief description of some classification algorithms is provided.

• Support Vector Machines

Support Vector Machines (SVMs) are a set of supervised machine learning algorithms, which are used in classification and regression problems. Each sample in the training data is plotted as a point in an n-dimensional space where n refers to a sample's features. The idea of the SVM algorithms is based on finding the optimal decision boundary (hyperplane), which will differentiate or separate the classes in a multidimensional space. The optimal hyperplane thus is the one that maximises the margin (also known as the distance between the classes). A wide range of kernel functions could be used for the Classifier

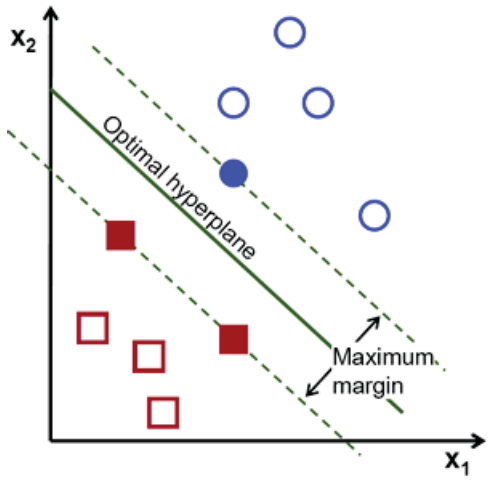


Figure 2.2: The optimal hyperplane with the maximum margin for an SVM trained on data belongs to two classes; adopted from [129].

implementation, such as radial basis function (RBF), linear and polynomial kernels. In the case of having a linearly separable training space, Support Vector Machine with linear kernel function could be used as shown in Figure 2.2 [25].

- **Decision trees**

In this type of classification a tree is constructed, based on the given training examples, where the constructed tree contains nodes, branches and leaves. The nodes of the tree represent the samples' features, while the branches represent the features' values; the class labels are then determined on each leaf. An example of a simple decision tree is shown in Figure 2.3. Therefore, when a new instance is seen and a label needs to be determined, the classification task will start at the root node moving down the tree through the corresponding branch, which holds the same value as the new instance. The process is then repeated at each node, until a label is reached on one of the leaves [138].

- **Ensemble Methods**

Ensemble methods are machine learning algorithms which construct multiple hypotheses (classifiers) that are used to classify the new data instances by outputting a weighted vote of their predictions. Such methods are known to produce more accurate results than

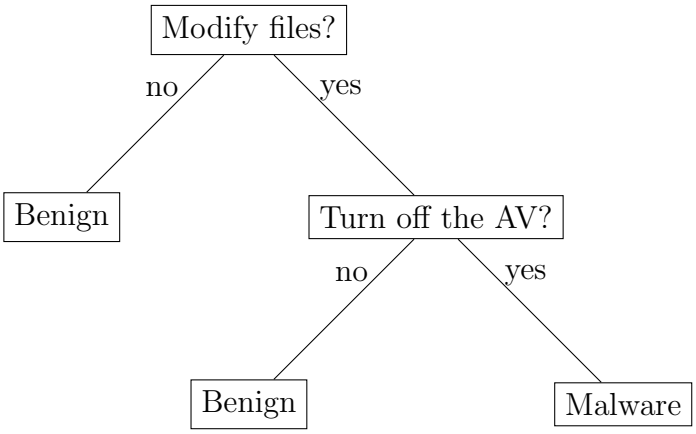


Figure 2.3: A simple decision tree.

the normal classifiers as a result of the averaged prediction [45]. The set of classifiers constructed during the training phase are referred to as *base classifiers*, which are generated from known learning algorithms such as decision trees, support vector machines and others. Two ensemble methods have been widely used in the area of malware detection to classify malware instances: *Bootstrap aggregating (Bagging)* [27], which is used to decrease the variance, and *Boosting* [146, 59] that is used to decrease the bias. Other ensemble methods (such as *Stacking*) are also used in machine learning literature to improve the overall classification predictions [197]; however, the difficulty of theoretically analysing the performance and results of such a method makes it less widely used than Bagging and Boosting [190]. In this thesis we focused on Bagging only as it was shown previously that such methods could generally outperform other ensemble methods [123], especially when dealing with an imbalanced dataset (where there is a significant difference between the number of instances in each class) [87]. The following section gives a brief overview of the Bagging method.

∗ **Bagging Classifiers:**

Bagging is a classifier ensemble technique that is based on randomly drawn subsets of the training data with replications (replacement), based on a uniform probability distribution. Each time a subset is drawn, a classifier will be constructed to classify the newly generated subset. The classification procedure is repeated a number of

times (for example 100 times) and a majority voting over all predictions is calculated as the final prediction to ensure the robustness of the results. In other words, given a training set T with size n, the bagging algorithm will generate a set of training sets T_i, by sampling examples uniformly and with replacement, where each of the newly-generated training sets is of size n' ($n' \leq n$). Bagging is usually used to reduce the variance of a prediction and to improve the stability and accuracy obtained by machine learning algorithms.

When classifying a dataset that has classes of an unequal sample size, some modifications could be integrated to the way that samples are drawn in Bagging. This is mainly because when the number of instances in each class varies (a.k.a having an **imbalanced dataset**) the classifier might become more biased towards the class which has the majority of the instances, and consequently the misclassified instances tend to belong to the minority class. Therefore, to decrease the effect of such a problem on the classification rate, various methods were suggested on the literature to balance the classes by either: decreasing the majority class (undersampling); replicating instances in the minority class (oversampling); or generating artificial data to increase the minority class (SMOTE) [35]. Studies also suggested combining such methods with Bagging, i.e. the entire minority class could be used as an input for each constructed classifier, along with a randomly generated subset of the majority class, which has the same size as the minority class, so balancing the data in each drawn subset. This method is referred to in machine learning literature as **Exactly Balanced Bagging (EBBag)** [34, 85, 87]. It was shown that such a method could actually outperform other similar classification techniques, especially if the subsets have been drawn without replacement [87], meaning that an instance could not be chosen twice in the same subset.

Different machine learning packages and libraries exist, which can be used to apply the proposed machine learning algorithms. Examples of these libraries are *WEKA* [196] and *Scikit-learn* [149].

2.3.2 Confusion Matrix Terminology and Evaluation Metrics

A confusion matrix is a table which is used to evaluate the performance of a classifier or a classification algorithm. By constructing such a table, the correctly and incorrectly classified instances are recorded; the calculation can then be used in further computations and analysis. Table 2.1 below shows an abstract example of a confusion matrix, with the following metrics (where the positive class is malware):

- True positive (TP): Number of positive samples correctly labeled as positive.
- True negative (TN): Number of negative samples correctly labeled as negative.
- False positive (FP): Number of negative samples incorrectly labeled as positive.
- False negative (FN) Number of positive samples incorrectly labeled as negative.

		True Classes	
		Positive	**Negative**
Predicted Classes	**Positive**	TP	FP
	Negative	FN	TN

Table 2.1: A Confusion Matrix computed for a two-class problem.

Given the previous terms, the accuracy measure of a classifier is given as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Where accuracy is the most commonly used metric, however, when having an imbalanced dataset (as discussed in the previous section) the normal accuracy presented in Equation 2.1 could be affected by the imbalance class problem. Therefore, other performance metrics have been proposed in the ML literature to measure the performance of a classifier in the presence of an imbalanced dataset. Examples of such metrics are the Geometric mean (G-mean) [99, 85] and the Area under the receiver operating characteristic curve (i.e. area under the ROC curve) [77].

2.3.3 ML-based Malware Detection Systems

As mentioned earlier in Section 2.2, machine learning algorithms can be greatly beneficial in the field of malware detection; they can be utilised by behaviour-based malware detection systems to help in identifying new malware threats. In this section, we review some of the proposed detection systems which can help in understanding the work presented in this thesis.

Schultz et al. [148] were the first to study the use of ML techniques to detect malware. They used three different methods to extract static features. The first method was binary profiling, in which for each binary they generated three different binary vectors: a list of the Dynamically Linked Libraries (DLL) that the binary called; a list of the functions that have been called within each DLL; and the number of functions called within each DLL. The remaining two methods of feature extraction were byte sequence and strings. They paired each of the described extraction methods to a specific learning algorithm, using a RIPPER rule based learner in order to extract rules from the binary profiling features; a Naive Bayes classifier to classify binaries based on the binary strings, and an ensemble classifier for the binary byte sequence. Of the three learning algorithms tested, the ensemble classifier achieved the highest detection rate at 97.76%, and when compared to traditional signature-based detection systems, it was able to achieve double the detection rate.

Since then, many studies have introduced or evaluated different ML-based detection systems. For example, the authors in [200, 203, 188] all based their detection systems on API call sequences extracted using static analysis. However, as mentioned in Section 2.1, static analysis is known to be questionable as a method for dealing with obfuscation techniques, as the sequence of the instructions might be modified and data locations can be hidden [115]. Motivated by this, other researchers have based their detection systems on API call sequences extracted using dynamic analysis [209, 176, 54]. Other behavioural features have also been used to detect malware using ML-based detection approaches, such as in Rieck et al. [141], where they proposed a method that involves learning the char-

acteristic features of malware and their typical behavioural pattern, which in turn helps to distinguish each malware family. Their proposed method consisted of three phases. In the first phase, malware behaviour was monitored, and analysis reports were produced using CWSandbox. Then, the resulting reports were embedded in high-dimensional vector space models, in which each represents a specific behavioural pattern. Afterwards, an SVM classifier was trained, based on the generated malware model, after labelling the data using the Avira Antivirus engine. In the final phase, distinctive features of each behavioural pattern were ranked for use in the classification of decisions. Their method reported 100% detection accuracy, when classifying known malware families against benign software. In addition, the method could predict nearly 70% of unknown malware from the total labels that were assigned by the Avira engine a month later. Rieck et al. [142] extended the previous method by introducing a prototype-based framework combining both classifying and clustering methods, in order to identify known malware families, as well as to discover novel malware classes. In the classification phase, the nearest prototype classification method was used, whereas in the clustering, the method used was complete-linkage clustering. The main feature of this framework is that it permits an incremental malware analysis, so the results of the previous run can be used to determine the decisions made about the current run, unlike most of the other frameworks proposed, which are based on batch analysis.

Firdausi et al. [56] compared the detection rate of multiple classifiers, by applying different ML algorithms to the same set of dynamic reports generated by Anubis Sandbox and embedded in vector space models. The algorithms used were k-Nearest Neighbours (kNN), Naive Bayes, J48 Decision Tree, Support Vector Machine (SVM), and Multilayer Perceptron Neural Network (MLPNN). The study concluded by highlighting that the highest detection rate was that obtained by the J48 decision Tree, with an accuracy of 96.8%.

It can be seen that detection systems, which are based on ML techniques are superior in terms of their ability to detect even those samples which have not been inspected before.

This is unlike traditional antivirus solutions, where the samples need to be examined and a signature needs to be determined in order to be able to detect malware. However, a misclassification could occur on ML-based detection systems due to the presence of noisy data, or due to changes in the underlying distribution of the features [208]. The latter changes are due to the fact that malware are evolving and their behaviour could change over time. The next section discusses briefly the nature of malware evolution, and their effect on machine learning-based detection systems.

2.3.4 The Effect of Malware Evolution on ML-based Detection Systems

Since the first malware emerged, there has been a continuous battle between security researchers and malware writers, where the latter are constantly evolving and trying to evade detection by adopting new malicious techniques. The evolution in malware characteristics (or features) over time has led to a non-stationary population, which means that features change over time. This phenomenon is commonly known in the ML and data mining context as *concept drift* [193, 178, 156, 2]. Machine learning-based detection systems, such as those described in Subsection 2.3.3, could be greatly affected by such a change, where the model used to classify malware becomes inconsistent with the new data that has arrived [156].

This problem was the motivation for proposing techniques which aim to track and detect changes in malware detection [156, 84], and to adapt when a change is detected [121]. Other researchers have also examined the performance of such detection systems over time [80, 156, 151], and checked whether a classifier could continue providing a good detection rate, or whether a retraining of the classifier was required; as a significant decrease in the classifier performance could be an indicator of concept drift [90, 156].

Although a large number of works have tracked changes in malware, proposed ways to adapt when changes are detected, and tried to determine when a new learning model needs to be constructed, we found that little attention has been paid to identifying reasons

that led to the misclassification of some malware samples, and that led to the drop in the detection rate in those systems. Identifying the reasons behind the misclassification of malware could result in a better understanding of malware behaviour, and the resulting misclassification rate, thus determining the optimal mitigation method. This problem has been addressed in Chapter 4, where a typical implementation of a machine learning based malware detection system has been used to classify malware samples, and analysis of the resulting misclassification rate have been conducted.

2.4 Cloud computing and Virtualisation technology

Before discussing the problem of malware in the cloud, a brief but essential overview of the cloud is presented.

Cloud Computing can be defined as a pool of dynamically scalable virtualised computing resources that could be accessed on demand over the internet; it thus offers, to users and organisations, significant economical benefits compared to traditional computing [58]. Cloud infrastructure relies on virtualisation technology which allows dividing the resources between multiple instances of virtual machines (VMs) that, in turn, results in the efficient use of the existing computing resources [195]. Each virtual machine could be described as a software implementation of a physical computer system that is hosted by a native machine and shares its resources to run programs like a real computer system [24].

The host program that is responsible for creating, managing and running VMs and dividing the resources between the VMs created is the Virtual Machine Monitor (VMM) or the so-called the *Hypervisor*. It can be considered as a light-weight operating system that is responsible of controlling the hardware resources and making them available for the implemented VMs when needed. In general, VMMs have two main functionalities which can be highlighted as follows [177]:

- Offering an isolated environment: where all communications between the underlying hardware resources, and the VMs can go through the VMMs. Therefore, every

VM will run as a separate physical machine, and it will not be affected by any configuration issues or by the instability of the other VMs that share the same resources.

- Managing the available resources: the VMM can manage and share the hardware resources between the implemented VMs, where every single physical resources can be mapped to multiple logical representations. Most importantly, the structure of these logical partitions is subject to change as demand varies.

According to *the National Institute of Standards and Technology* (NIST) in [106], the cloud can be deployed in four different models, namely, private, community, public and hybrid clouds; the deployment model depends on the group of users who will benefit from the cloud resources and services offered. In the private clouds, the services are offered to be exclusively used by a single organisation. They can be managed by the organisation, by a third party, or by a combination of both; and they can be deployed on or off the organisation's premises. In the community cloud, the cloud services and functionality are shared among a number of organisations which have common concerns or goals and it is not limited to one organisation only as in the case of the private cloud. It can also be deployed in one of the organisations or off the organisations' premisses; and it can be managed by one or more of the organisations, or by a third party. A public cloud, on the other hand, is managed by the cloud provider and it offers the cloud services and resources to the public where they can choose between short or long-term contracts [64]. In the case of the hybrid cloud deployment model, a combination of the first two cloud models (i.e. private and public) can be obtained in a way that ensures the isolation of the different cloud entities while facilitating the data and application portability. The shared and multi-tenant nature of the public cloud model makes it less secure than the private cloud model, since the resources provided and access are not restricted to a limited number of tenants, as in the case of a private cloud, which is considered the most secure model [157].

In addition to the different deployment models, the cloud can also be classified in

terms of the services it offers to consumers in one of three different categories: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Minimum control is offered through the SaaS service model, where the consumer can access different applications offered by the cloud provider and use them, but without having any control over the underlying infrastructure, or even over the application used. On the other hand, consumers will have full control when using the IaaS service model, as they can deploy their own operating system on the hosted VMs, but without having control over the underlying cloud infrastructure. The flexibility provided by IaaS comes at a cost as the VM tenants need to maintain an upgraded system and fully patched applications; such a service model is therefore considered as more vulnerable to attacks compared to other service models.

2.4.1 Malware Detection in the Cloud

The risk of malware infection is not limited to classical computing only; cloud computing and its Virtual Machines are also vulnerable to such a threat, where the security level provided depends on the deployment and service model requested. The impact of an infection in the cloud environment could potentially be even greater. When a VM is compromised by exploiting a vulnerability (e.g. VENOM [42]), the damage might spread to the entire underlying infrastructure, which hosts not only that VM, but many others; a large number of VMs can then be used to initiate various malicious activities, such as Distributed Denial of Service (DDOS) attacks and the sending of spam [18]. This threat is a well-recognised risk to the cloud community and it was mentioned by NIST in [74].

The configuration of a detection system in a virtual environment can conform to one of two dominant cases, each of which has its own advantages and disadvantages. In the first case, the detection system is installed on the system that hosts the VMs. Advantages provided by such an approach are: (i) reducing the required memory and CPU; and (ii) offering the necessary isolation between the infected system and the malware scanner, thus reducing the possibility of the scanner being compromised. However, the detection

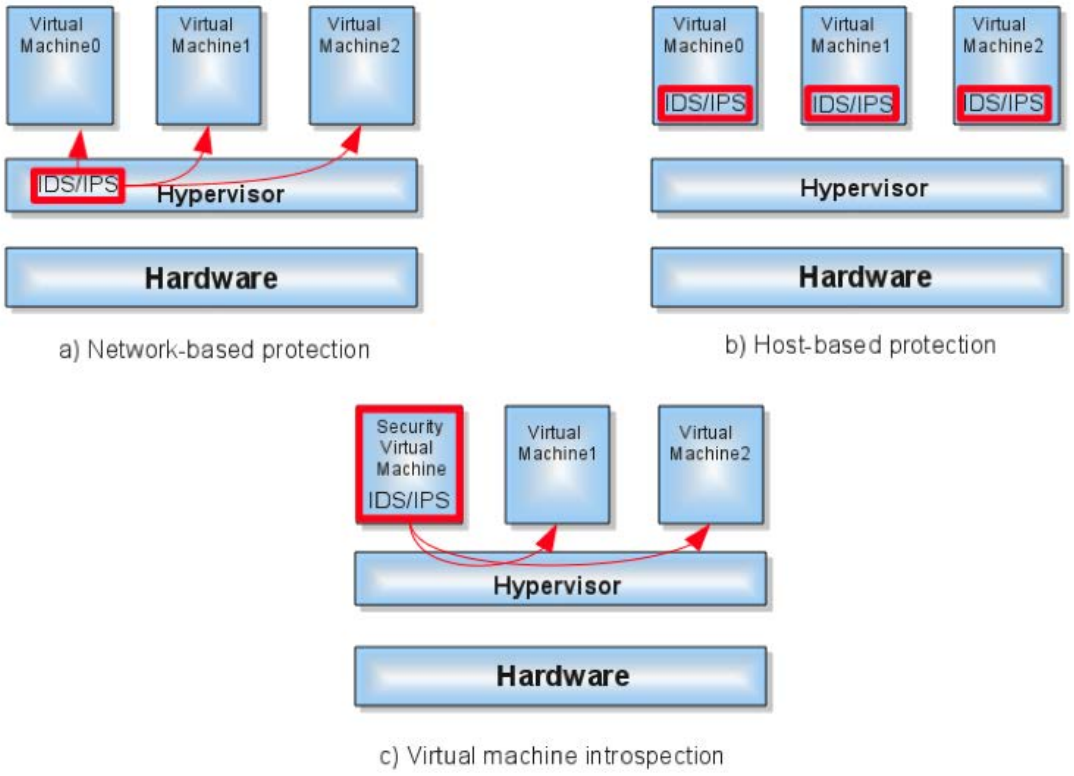


Figure 2.4: VM protection approaches, adopted from [177]

system in this case would have a limited view over the monitored VMs. The second case solves the limited context problem by installing a detection system in each VM; thus a full view over the inspected VMs can be guaranteed. However, two issues may arise in this case: (i) the CPU and memory consumption in such an approach tends to be very high; and (ii) the detection system might be susceptible to infection and manipulation as it is not isolated from the infected system [63].

As a result, several studies have been conducted with the aim of balancing the trade-off between security enhancement on one hand and the comprehensive view and the efficient usage of resources on the other hand. In fact, this was the motivation for introducing the *Virtual Machine Introspection technique* (VMI), which refers to the process of monitoring other VMs by checking their memory pages [63]. The novelty of this technique is that it resolves the known trade-off between the isolation and the comprehensive view during the inspection, [48] in the way that it benefits from the hardware virtualisation in the cloud,

to provide an out-of-the-guest malware analysis and detection. The monitoring process is carried out by the VMMs or other VMs which are assigned a special privilege. Thus, instead of having the security software within the customer's VM, such a system could be drawn outside the VM. This will give it a good overview of the inspected VM, while rendering it invulnerable to potential attacks.

VMI has been used in developing a wide range of services and tools such as, cloud's IDS [92, 86], malware detectors [79, 155], malware analysis tools [46], and digital forensics [20, 1]. Further research has also been carried out in order to provide introspection-as-a service in the cloud through introducing written libraries and tools [192, 98, 11]. Other studies have also proposed VMI-based solutions in the form of in-cloud services to serve as light-weight agent to end users connected to the internet [125, 126, 70, 3]; however, these solutions are outside of the scope of this thesis.

The following section provides an overview of the VMI-based malware detection techniques in the cloud environment.

2.4.2 VMI-based Malware Detection Systems

Garfinkel et al. [63] were the first to introduce Virtual Machine Introspection-based detection systems. They proposed an Intrusion Detection System (IDS), referred to as Livewire, which was built as a customised version of VMware Workstation for Linux x86. Somewhat later, Kourai and Chiba [92] introduced an IDS called the HyperSpector which targets distributed computer systems. The implemented IDS benefits from VMI technology in isolating the server that it monitors from the IDS to reduce the cost of implementing an IDS that is physically separated from its hosts.

Since that time, different VMI-based intrusion, and malware detection systems have been proposed, some of which have relied on guest cooperation, whereby agents or modules are installed inside the VMs to monitor the guest, and to facilitate the inspection [133, 153, 79, 111, 185]. However, such an approach is susceptible to manipulation by an attacker, because the malware detector and the protected VM are not isolated from one

another [174].

Another type of VMI-based technique is the external approach, where no agent is installed within the protected guest [83]. According to Nance et al. [120], VMI-based techniques can also be classified according to the introspection method, either as state introspection (passive monitoring), or event introspection (active monitoring). In **active monitoring**, several predefined events are tracked in order to facilitate interfering with the threat, rather than being limited to only detecting and reporting the threat. However, as the state of the inspected VM could be altered in this type of introspection, a compromised VMI-based tool might undermine the security and integrity of the inspected VM [154]. In **passive monitoring**, on the other hand, VMI techniques and tools can be used to monitor the state of a VM periodically, thus detecting changes that might arise due to potential malware infections. Examples of such malware monitoring approaches include Livewire[63], VMwatcher[83], Virtuoso[47], Forensic Virtual Machines[69], and Bitdefender Hypervisor Introspection (HVI)[22]. For the sake of convenience, we will be using the phrase *VMI-based monitoring approaches* in the following chapters of this thesis to refer to those approaches that perform *passive* monitoring of the targeted VMs as the focus will be on those techniques only.

Forensic Virtual Machines (FVMs)

Besides reducing the attack vector by using passive VMI-based approaches to monitor the VMs, the monitoring task could be subdivided among a number of small scanners or mini-VMs (referred to as Forensic Virtual Machines), in order to reduce also the scanning overhead [69]. An FVM is dedicated to identifying the existence of a single symptom given an assigned time interval which is referred to as Permissible Time to Stay (PT2S); if the end of this time is reached and the FVM did not find the symptom that it was looking for, then it will move to a new VM target to scan it. In addition, an FVM is shared among a number of VMs; therefore, each FVM is initialised with a special distributed algorithm (referred to as a *mobility algorithm*), which is responsible for scheduling the movement

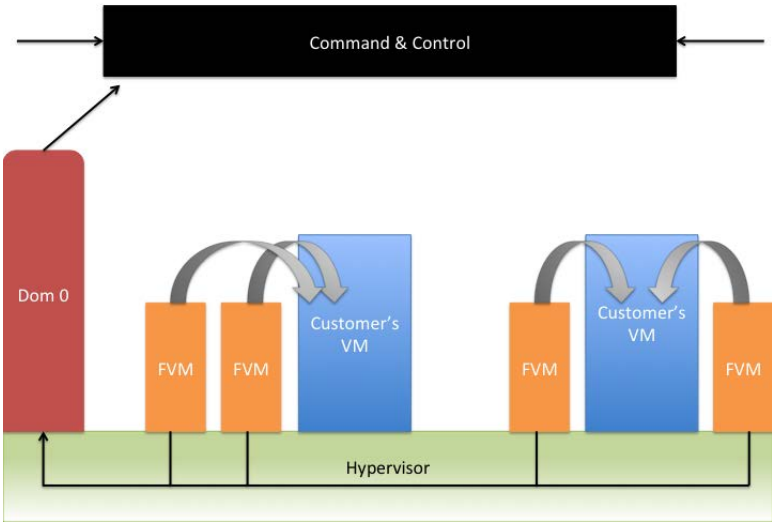


Figure 2.5: FVMs inspecting customers' VMs, adapted from [69].

of the FVM from one VM to another. Figure 2.5 shows a number of FVMs, given the capability to inspect the memory pages of customers' VMs.

The internal structure of an FVM consists of three components: notification, mobility and search. The *notification* module is responsible for reporting a symptom that has been inspected (either found or not found); the *mobility* module is the one responsible for scheduling the movement from one VM to another as FVMs can be shared among many VMs. The mobility algorithm embedded could vary from simple to dynamic algorithms such as random movement, scanning according to a pre-determined order, or based on advanced scanning algorithms (e.g. [69, 154, 9]). The *search* module allows switching between a number of different search strategies; some of these are designed to search within the registry while others can examine the process table [155].

An observation that can be made here is that when using a malware detector on an open networked environment, such as a cloud environment, the need to identify an infection as soon as possible is vital; therefore, using light scanners with a low usage of resources such as FVMs can be beneficial. As mentioned previously, they also provide the required isolation between the malware scanner and the infected system, making it harder for the scanners to be compromised. However, to achieve the full potential of such

malware scanners, it is important to identify the most efficient mobility algorithm prior to the scan, which could boost the efficiency of the scanning process, as the creation of FVMs is a resource-intensive task. This challenge has been addressed in Chapter 5, where a probabilistic approach was proposed to identify the most beneficial mobility algorithm that could boost the efficiency of the FVMs used.

Another observation is that the minimum performance impact on the cloud's VMs which results from using VMI-based malware detection methods (e.g. FVMs) could be at the expense of wrongly flagging more VMs as infected. This is because VMI-based techniques are based on identifying symptoms that exist in both malicious and normal behaviour, but in different proportions or combinations; where in some cases a clear-cut distinction between the two behaviours is not easy to identify; thus they can provide a means for detecting early signs of infections but without providing definitive identification results. Unlike signature-based methods, which use a fingerprint-like signature to identify malware threats with an extremely low false positive rate [67], but with the high usage of the VMs resources during the scan [150]. This issue, of balancing the trade-off between the accurate identification of infection and the consumption of VMs' resources, has been addressed in Chapter 6, where a market-based prioritisation approach that combines light-weight and heavy-weight malware detection approaches has been proposed to allocate the heavy-weight scanning resources wisely to the VMs which need them the most; thereby resulting in a wise use of resources while reducing the false alarms generated.

2.5 Probability Theory and Bayes' Theorem

Probability theory is the branch of mathematics that is concerned with experiments; it measures the likelihood that a specific event will occur. In the field of malware detection, the experiments can refer to the execution of the binary in a controlled environment, while the event represents the occurrence of malicious behaviour (the occurrence of a group of malicious symptoms) during the execution of that binary. An example of a

single symptom is modifying the registry key to disable the firewall, or creating a new directory for the malicious file. In an experiment, the set of all possible outcomes o is referred to as the experiment's *sample space* Ω; here we are only interested in discrete sample spaces. Definitions and theorems mentioned in this section could be found in any introductory book on probability theory (e.g. [41]).

The probability function P should satisfy the following fundamental axioms (known as Kolmogorov's axioms of probability):

Definition 2.1 (Kolmogorov's axioms)**.**

Axiom 1: *(Non-negativity) The probability of an event A occurring is a real, non-negative number:*

$P(A) \in \mathbb{R} \qquad P(A) \geq 0 \qquad \forall A \subseteq \Omega.$

Axiom 2: *(Normalisation) The probability of the entire sample space Ω is unity:* $P(\Omega) = 1.$

Axiom 3: *(Additivity) For any sequence of events $A_1, A_2, A_3, ...A_n$ which are mutually exclusive, the probability of all the events equal the sum of the probabilities of each of those events:* $P(A_1 \bigcup A_2 \bigcup A_3 \cdots \bigcup A_n) = \sum_{i=1}^{n} P(A_i)$

As consequences of Kolmogorov's axioms of probability, the following applies:

Theorem 2.1 (Consequences of Kolmogorov's axioms)**.**

1. $P(\varnothing) = 0.$

2. $A_1 \subseteq A_2 \Rightarrow P(A_1) \leq P(A_2).$

3. $P(A) \leq 1.$

4.If $\neg A$ and A are Complementary Events in the sample space Ω, then $P(\neg A) = 1 - P(A).$

As we have multiple symptoms that can occur during the execution of malware, we are often concerned with the probability of an event occurring given that another event had been already observed, this leads to the definition of conditional probability which

can be written formally as follows:

Definition 2.2 (Conditional probability)**.**

For two discrete events A and $B \subseteq \Omega$, the conditional probability of the event B given that event A has occurred, can be given by:

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \qquad P(A) \neq 0$$

As a consequence of this definition, the multiplication rule of probability can be defined as follows:

Theorem 2.2 (Multiplication rule of probability)**.**

For two discrete events A and $B \subseteq \Omega$, the probability of them occurring together can be given by:

$$P(A \cap B) = P(B|A) \cdot P(A)$$

It can be also written as follows when $P(B) \neq 0$

$$P(A \cap B) = P(A|B) \cdot P(B)$$

2.5.1 Bayes Theorem

Bayes' theorem was originally developed by Thomas Bayes; it can be used to describe how an initial belief could be updated according to new evidence observed. The theorem is written formally as follows [128] (derived using Theorem 2.2):

Theorem 2.3 (Bayes theorem)**.**

If $A \subseteq \Omega$, and $B \subseteq \Omega$ are two discrete events, and $P(B) \neq 0$, then:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

- $P(A)$ is the prior probability of having the initial belief or hypothesis A.
- $P(A|B)$ is the posterior probability of a hypothesis A, given that evidence B occurred.
- $P(B|A)$ is the likelihood of evidence B on hypothesis A.
- $P(B)$ is the probability of observing the evidence B.

For two mutually exclusive and exhaustive events A and $\neg A$, $P(B)$ can be expanded to the following:

$$P(B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A) \tag{2.2}$$

Therefore, by using Equation 2.2, Theorem 2.3 can be rewritten as follows:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)} \tag{2.3}$$

2.6 Information Theory

Information theory is the branch of mathematics that studies uncertainty and different information-related measurements and quantities; for example, it provides ways to measure and quantify the information shared between two random variables. In this thesis, we are focusing only on the quantities which are required to understand our contributions to the literature. Definitions and theorems mentioned in this section could be found in any introductory book on information theory (e.g. [41] and [152]).

2.6.1 Mutual Information (MI)

In the field of information theory, the mutual information (MI) of two discrete random variables X and Y is the amount of information which can be learnt about one variable by observing the other. In other words, it is the amount of information that is shared

between the two random variables X and Y measured in *bits*. Mutual information (MI) can be defined mathematically as follows:

Definition 2.3 (Mutual Information)**.**

The Mutual Information (MI) between two random variables X and Y is given by:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \cdot log \ \frac{P(x,y)}{P(x)P(y)}$$

However, in some cases, the mutual information between the two variables X and Y needs to be obtained given that we have observed a third random variable or event $Z = z$, this can be given by:

Definition 2.4 (Mutual Information conditioned on Z=z)**.**

The Mutual Information (MI) between two random variables X and Y conditioned on a third random variable Z=z has occurred is given by:

$$I(X;Y|Z=z) = \sum_{y \in Y} \sum_{x \in X} P(x,y|z) \cdot log \ \frac{P(x,y|z)}{P(x|z)P(y|z)}$$

In the case of conditioning on all the events $z \in Z$, we end up with the definition of the Conditional Mutual Information.

Definition 2.5 (Conditional Mutual Information)**.**

The Conditional Mutual Information (CMI) between two random variables X and Y given that a third random variable Z has been observed is given by:

$$I(X;Y|Z) = \sum_{z \in Z} \sum_{y \in Y} \sum_{x \in X} P(x,y,z) \cdot log \ \frac{P(z)P(x,y,z)}{P(x,z)P(y,z)}$$

2.7 Market Mechanisms

In this section, we provide a brief, but essential introduction to market mechanisms. This introduction is needed to understand the contribution made in Chapter 6, where a market-based approach has been proposed for the efficient and effective detection of malware infections in the cloud via introspection.

Computer systems adopt market mechanisms in order to define new ways to allocate the services or the resources in a shared and decentralised environment such as the cloud to a number of tasks. This is based on the fact that those tasks that need to be fulfilled are not of equal priority; therefore, using methods which are influenced by market mechanisms in such situations helps in prioritising the tasks based on a dynamic price assigned to each of them during the allocation process [119]. In this chapter, the allocation is concerned with assigning a number of malware scanners to a different number of VMs; their need to be scanned can vary based on the symptoms discovered on each of them.

From an economic point of view, a market refers to the place in which goods and services could be bought and sold [179]. Sellers and buyers[1] are examples of market agents who carry out their trading based on their own valuation of a good; the decentralised characteristic of the market results from the competitive and self-interested nature of its agents [130, 39]. According to Clearwater [38], systems which are based on a market mechanism or a number of the market-specific features such as their decentralised characteristics, the interactions between the different agents, or the notion of the resources or services allocated; they are referred to as market-based systems.

The set of rules and protocols used, which define the trading interaction between the market agents, to match the goods offered to the agent who needs them the most, is known as the pricing policy or mechanism. It can be categorised, broadly, into two classes: posted-price mechanisms, and price-discovery mechanisms. According to Elmaghraby and Keskinocak [51], the difference between the two categories, that is in a posted-price

[1]Sellers and buyers in the current context refer to the set of FVMs available and the VMs that request to be scanned, respectively.

mechanism, take-it-or-leave-it predetermined prices are offered to sell the goods; on the other hand, in a price-discovery mechanism the prices of goods are determined during the transaction, based on a bidding process or an auction. Auctions can offer a dynamic pricing alternative to the traditional posted-pricing mechanism, when there are no fixed prices; or where prices cannot be pre-determined for the goods that are offered [32]. In this thesis we are using a price-discovery mechanism (represented as a bidding process) to allocate the available FVM instances to the VMs which need them the most; in other words, we are using the symptoms discovered in each VM at any given time to determine the criticality of the VM, thus estimating its need for a scan, in addition to determining the scanning type that is needed (a light inspection by the available FVMs or a full scan).

Decisions in auctions can be determined based on simultaneous or sequential bidding. In simultaneous bidding, bids are submitted only once, and the prices and allocations are determined immediately; an example is sealed bid auctions. In sequential bidding, the bidding process involves various rounds of price and demand alterations, or consecutive price adjustments, with a pre-defined halting rule [94]. In this thesis we are only concerned with sealed bid auctions because decisions are made instantaneously, resulting in an efficient allocation of the scanning resources, without causing a considerable overhead during the allocation process [66].

2.8 Conclusion

This chapter provided background knowledge and reviewed several related notions. It started by presenting the main analysis types used to analyse malware (i.e. static and dynamic analysis). It discussed also the main methods of detecting malware (signature-based and behaviour-based malware detection methods), and also presented several techniques utilised by these detection methods (i.e. ML-based and VMI-based malware detection techniques). The chapter has also looked at some background and definitions that are necessary to follow the work and contributions made, such as the virtualisation technol-

ogy and the cloud, a number of mathematical theories (i.e. probability and information theory), and market mechanisms.

Before addressing the main research questions posed in Section 1.2, a representative malware corpus has to be developed in order to facilitate conducting the experiments needed to answer the research questions previously mentioned, and to evaluate the approaches presented in this thesis. Therefore, the next chapter presents the process of collecting, analysing malware, and extracting the required features to be used in the main chapters of this thesis.

CHAPTER 3

DEVELOPING A FINE-GRAINED MALWARE FEATURE SET

3.1 Introduction and Motivation

Malware features (or symptoms) are the main source of information used for implementing behaviour-based malware detection systems. Collecting a representative malware corpus, and encompassing a variety of malware types, families and variants is essential, as this will result in generating a rich and varied feature set, which can enhance the detection process and covers a large number of malicious activities. This chapter discusses the approach adopted to develop a fine-grained malware feature set that was used in the subsequent main contribution chapters of this thesis. In addition, this chapter outlines the process of developing an abstraction of the behaviour of the samples collected, which is represented in the textual logs documented during the analysis process. This chapter also provides some overall statistics regarding the knowledge-base generated.

The chapter's contents are based partially on two of our papers, Alruhaily et al. [6] and Alruhaily et al. [7]. It is important to note that the contents of this chapter forms an important source of information for this thesis, as all the works presented were possible, and have been evaluated using features from the knowledge-base described in the following sections.

3.2 Sketch of the Approach

In order to achieve the objectives of this research, it was essential to develop a knowledge-base, which provides the set of symptoms that is expected to be seen in terms of both, malicious and normal activities. We thus developed a framework that automates the process of downloading malware samples, analysing and storing the raw features generated for each of the samples collected. Figure 3.1 depicts a high-level overview of the framework, where the *Python* programming language was used for the implementation.

In summary, the implemented framework can be broken down into multiple components that perform the following tasks:

1. Raw data collection: in this phase malware samples are downloaded using two approaches: randomly and based on their family and variant name. This process is described in detail in Section 3.3.

2. Feature generation and extraction , which includes:

    - Behaviour monitoring: in this phase, the malware and benign samples were analysed with two widely known malware analysis tools: Anubis and Cuckoo sandbox, in order to collect the raw features to be used for further analysis and investigation; this process is described in Subsection 3.4.1.

    - Feature transformation: this includes transferring the raw data collections generated in the previous step into a representative set of features to distinguish malware from benign samples. The details of this process are documented in Subsection 3.4.2.

These tasks are described in detail in the following sections, where each section delivers an overview of the procedure followed to fulfil the main goal of each task.

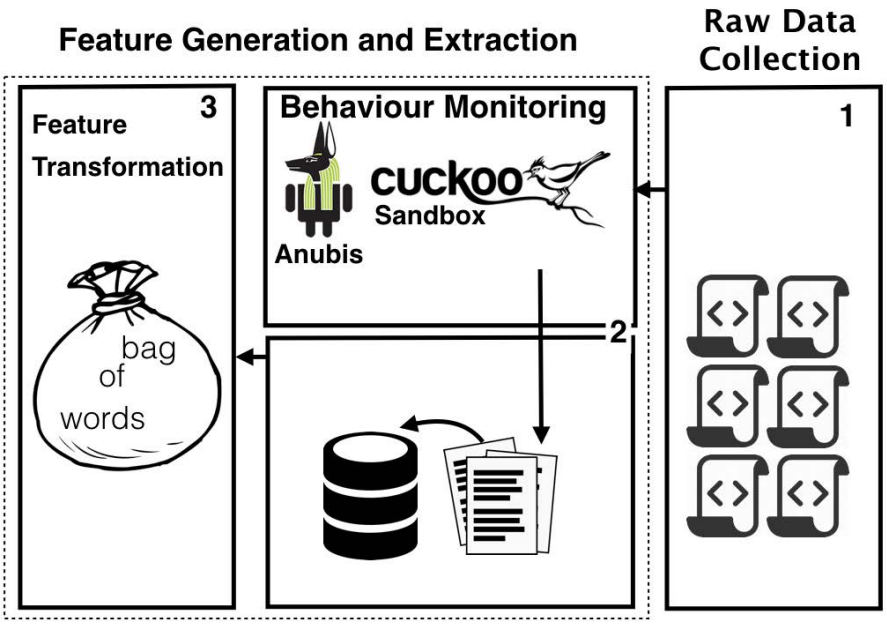


Figure 3.1: Downloading and analysing malware samples.

3.3 Raw Data Collection

Initially, the Python-based tools: Mwcrawler and Maltrieve [103, 104] were used to collect malware samples from a number of sources. This involved parsing multiple malware websites and retrieving the latest malicious samples uploaded to them. These websites include: Malc0de, Malware Black List, Malware Domain List, Malware URLs, VX Vault URLquery, CleanMX and ZeusTracker.

To ensure that the dataset reflected the most common types of malware, we developed a Python script that collected the top 10 and 20 malware families recorded by Symantec and Microsoft in their *Internet Security Threat* reports [166], and *Security Intelligence* reports [110], respectively. The script works by pulling all samples resulting from the search request for each malware family, including all the available variants, from an open malware database (i.e. Open Malware [127]). In addition, a large number of samples were also downloaded from the VirusTotal's website [184] through their *intelligence* service. We note that this method succeeded in getting more samples from the most common malware families. Overall, we have collected malware samples that varied across approx-

imately 800 malware variants, with the date of discovery varying between 08.12.1997 (*Infostealer*) [160] and 22.10.2015 (*W32.Xpiro.I*) [161]. During the samples collection process, we only considered malware and benign samples in the portable executable (PE) file format. Benign executables were collected from a fresh installation of Windows XP and several official websites; e.g. Microsoft, Adobe and others. All the executables were sent to the Virustotal [184] website to retrieve the scanning results from multiple vendors (e.g. Symantec, Kaspersky, Avira, ClamAV), and to ensure the integrity of the benign samples.

3.4 Feature Extraction

After samples have been collected, each needs to be analysed using static or dynamic analysis (as described in Chapter 2); thus, information regarding the behaviour of this sample could be recorded. The information collected at this stage can be viewed as raw data. In this thesis, we are concerned only with behaviour-based malware detection techniques and with analysing and extracting raw features of the malware using dynamic analysis techniques only.

When the analysis is complete, the required input can be extracted from the raw data documented; it would then need to be transformed into a reduced representation, so that each sample in the dataset could be represented by a set of features. In malware detection tasks, different type of features could be used to represent binary files; the most common types of features include:

- Application Programming Interface (API): the Windows API describes a set of functions and routines provided by the Windows operating system, which can be used by a programmer when developing a Windows application. These functions offer access to a number of services to facilitate interacting with the operating system. Thus, monitoring these functions' calls and their parameters can give a good representation of the behaviour of any application including malware [52].

- Samples' actions: This can be documented by some analysis tools, such as Anubis. The actions recorded include opening or reading a file; creating, deleting or opening a registry key; connecting to the internet or creating a mutex [14].

- Operational Codes (or Opcodes): These refer to the portion of a *machine language instruction* that specifies the operation to be performed [210]; examples include, mov, push, call, pop ... etc.

The process of extracting and transforming the raw features is described in Subsections 3.4.1 and 3.4.2, respectively.

3.4.1 Behaviour Monitoring

In behavioural monitoring, one or more dynamic analysis tools are used to obtain a detailed trace of the actions the suspicious file performs during its execution. The principal aim of this procedure is to extract a set of features which can then be used as input for the malware detection process. We used two well known (as shown in the literature) malware analysis tools: an open-source, local-based sandbox (i.e. Cuckoo sandbox) and an online-based sandbox (i.e. Anubis). The following subsections present an overview of these two tools.

Anubis

Anubis is an online-based dynamic analysis tool, which runs binaries for 4 minutes in an emulated PC environment in which Windows XP SP3 is installed. The logs generated by Anubis are saved in XML, HTML and TXT formats, and they contain system actions and network activities, such as creating, modifying or deleting a file; reading, writing, or creating a registry key or value, and loading a Dynamic-link library. In order to submit malicious and benign samples to Anubis, a Python script written by Anubis developers was used [10]; whereas we developed a script to retrieve XML reports after the analysis phase was completed. A snapshot of a report generated by Anubis can be seen in Listing 3.1.

Listing 3.1: A snapshot of a report generated by Anubis.

```
1 <reg_value_modified count="1" key="HKU\S
     -1-5-21-842925246-1425521274-308236825-500\Software\Microsoft\
     Windows\CurrentVersion\Policies\Explorer" value_data="0"
     value_name="NofolderOptions"/>
2 <reg_value_modified count="1" key="HKU\S
     -1-5-21-842925246-1425521274-308236825-500\Software\Microsoft\
     Windows\CurrentVersion\Policies\System" value_data="1" value_name=
     "DisableRegistryTools"/>
3 <reg_value_modified count="1" key="HKU\S
     -1-5-21-842925246-1425521274-308236825-500\Software\Microsoft\
     Windows\CurrentVersion\Policies\System" value_data="0" value_name=
     "DisableTaskMgr"/>
4 <reg_value_modified count="1" description="auto_start" key="HKU\S
     -1-5-21-842925246-1425521274-308236825-500\Software\Microsoft\
     Windows\CurrentVersion\Run" value_data="C:\WINDOWS\system32\regsvr
     .exe" value_name="Msn␣Messsenger"/>
5 <reg_value_read count="1" key="HKLM\SOFTWARE\CLASSES\.EXE" value_data
     ="exefile" value_name=""/>
```

Cuckoo Sandbox

Cuckoo Sandbox is an open source automated malware analysis system. The sandbox executes samples in a controlled environment (*Virtual Machines*) for 2 minutes, and generates reports in JSON, HTML and TXT formats, with API calls, system traces, network activities in .pcap file. To reduce the analysis time, Cuckoo Sandbox was configured to use three VMs instead of one where all will run simultaneously on a host machine running Ubuntu 14.04. Windows XP SP3 was selected to be installed on the VMs during the analysis phase, as all the malware samples collected, from 08.12.1997 up to 22.10.2015, can run on this system, according to the Symantec's *systems affected* information for each malware family. In addition, Windows XP SP3 is widely used to analyse malware [33, 135] for numerous reasons, including the fact that it demands less memory and CPU [135]. After the analysis stage is complete, the sandbox saves behavioural logs locally for further analysis of the executables. A snapshot of a report generated by Cuckoo Sandbox can be seen in Listing 3.2.

Once the analysis is completed and the reports with the raw logs are received, in

Listing 3.2: A snapshot of a report generated by Cuckoo Sandbox.

```
"process_path": "C:\\Documents␣and␣Settings\\nnnnnn\\Local␣Settings\\
   Temp\\002
   aba90c1034916ee59e3d93d8dd574c0d28319fdc1b0e3478bac5efd47a0c6.exe"
   ,
"calls": [
    {
        "category": "registry",
        "status": 0,
        "stacktrace": [],
        "last_error": 0,
        "nt_status": -1073741515,
        "api": "NtOpenKey",
        "return_value": 3221225524,
        "arguments": {
            "key_handle": "0x00000000",
            "desired_access": "0x80000000",
            "regkey": "HKEY_LOCAL_MACHINE\\Software\\Microsoft\\
               Windows␣NT\\CurrentVersion\\Image␣File␣Execution␣
               Options\\ntdll.dll"
        },
        "time": 1462563966.375,
        "tid": 748,
        "flags": {
            "desired_access": ""
        }
    },
```

either JSON or XML format, the reports are automatically parsed with the implemented parser, so that the documented actions of malicious or benign executables, and the other information recorded in the report can then be added to a local *Postgres* knowledge-base to ease the process of retrieving the required information. We also implemented a Python script to extract all the information found on the Symantec's *Threats, Risks & Vulnerabilities* [169] web pages to collect other valuable information, such as details regarding malware types, the discovery date of each malware family, the risk level and the systems affected.

Although the information detailed in the reports has been added to the database in order to ease the access and simplify the information query process; in cases where data are to be used as an input for a machine learning-based detection system, it needs to be

transformed into a feature vector, that is, a more suitable representation for the learning algorithm. This process is usually referred to as the *feature transformation* process. The following subsection describes this process in detail.

3.4.2 Feature Transformation

After parsing the analysis reports and processing all the raw information included, the data then needs to be transformed into a suitable set of features or vectors to support the learning and detection process. The process of transforming the raw features can be achieved through Tokenisation, and Vectorisation [26]. The following subsections describe those sub processes in detail.

- **Tokenisation**

  The raw data extracted are used to generate a list of strings of a given size (tokens), where each individual token is considered as a feature. The set of features at this stage is referred to as a *Bag-of-words* or as a *Bag of n-grams* representation. In the simplest form of the Bag-of-words (Bag of n-grams) model, a collection of single words (also known as unigrams) is produced. In this case, only the word counts or occurrence is important and little attention is paid to the order of the words; this additional information can be important, especially in a malware detection context, as the features' order represents the captured behaviour. Therefore, other n-gram sizes could be used to extract the tokens, where the token not only contains a single word, but also preserves sequences of two, three or more words; thus it can be considered as taking a snapshot of the malware behaviour.

- **Vectorisation**

  When passing the features (i.e. the extracted tokens) to a machine learning algorithm, the feature set needs to be converted further to a numeric representation, in order to generate feature vectors, in which different weights for n-grams could be computed, such as:

- Term presence, also called binary vector representation, where '0' refers to the absence of a specific token and '1' denotes its presence.
- Term frequency, refers to the number of times a token is found in a malware or benign sample.

In this thesis, according to the detection technique used, different malware features' types (discussed in Section 3.4), and feature transformation methods were used. These choices can be summarised briefly:

- In Chapter 4, due to the fact that the features will be used as an input for an ML-based detection system, we used the API calls, generated by Cuckoo Sandbox, to represent each executable as they offer a good representation of malware behaviour. We carried out a comprehensive investigation to identify the most suitable bag-of-n-grams representation; where unigram, bigram, a combination of unigram and bigram, and trigram representation have been evaluated. Using unigrams suggests that the sequence of API words is unimportant, where we simply check if a word is present or not. By using bigram and trigram, the feature vector will not only contain a single API word but also preserve the sequence of two and three words, respectively, which can be considered as taking a snapshot of the malware behaviour. We used the unigram+bigram as it showed that it can result in more generalisation compared to other n-gram sizes. The full details of the analysis has been documented in Subsection 4.3.3.

- In Chapters 5 and 6, instead of passing the behavioural vectors to the malware detector, individual features were used to detect malware. The features were broken down into tokens, where each represents a full registry path. The resulting features were used to identify the existence of a malware infection based on the registry paths accessed either, to modify, read, create or delete a specific registry key. The two Chapters mentioned used samples analysed by Anubis sandbox. Full details of the feature generation process in Chapter 5 and 6 is documented in Section 5.2

and Subsection 6.3.1, respectively.

3.5 Malware Statistics

As a result of the work carried out in this chapter, over 18,000 malware samples have been collected, of which 13,047 samples were analysed using Anubis, and 5,410 were analysed locally with Cuckoo Sandbox. The resulting knowledge-base contained different classes of malware, in addition to a number of samples with overlapping characteristics, as malware classes are known to be not mutually exclusive [62]. The percentage of each malware class available in the overall knowledge-base is shown in Figure 3.2. Moreover, Figure 3.3 shows the percentage of malware classes based on the dynamic analysis tool used to generate the behaviour. The distribution of malware types presented here is comparable to malware type distribution recorded by known security vendors during the same time period (e.g. [131, 132]). Table 3.1, on the other hand, shows the top 10 malware variants analysed by each analysis tool.

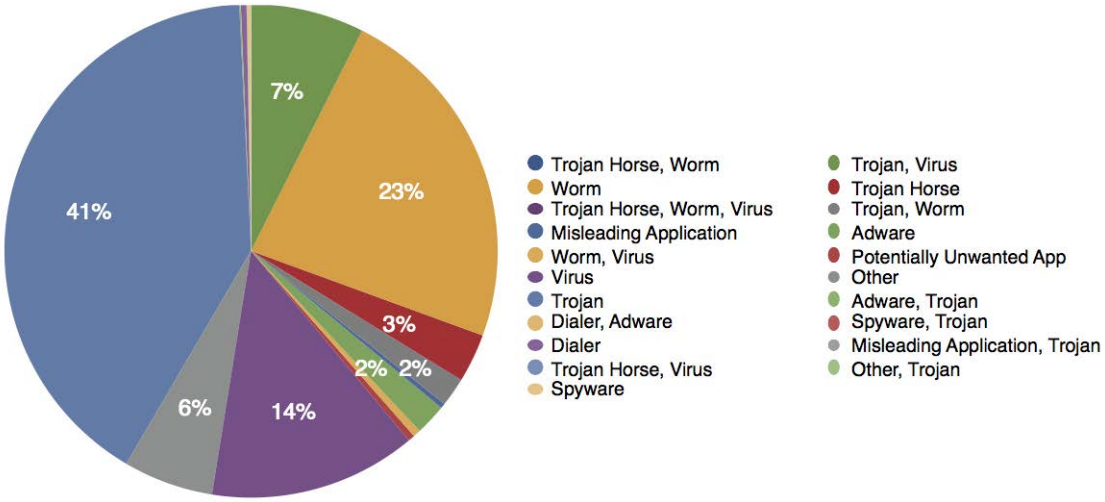
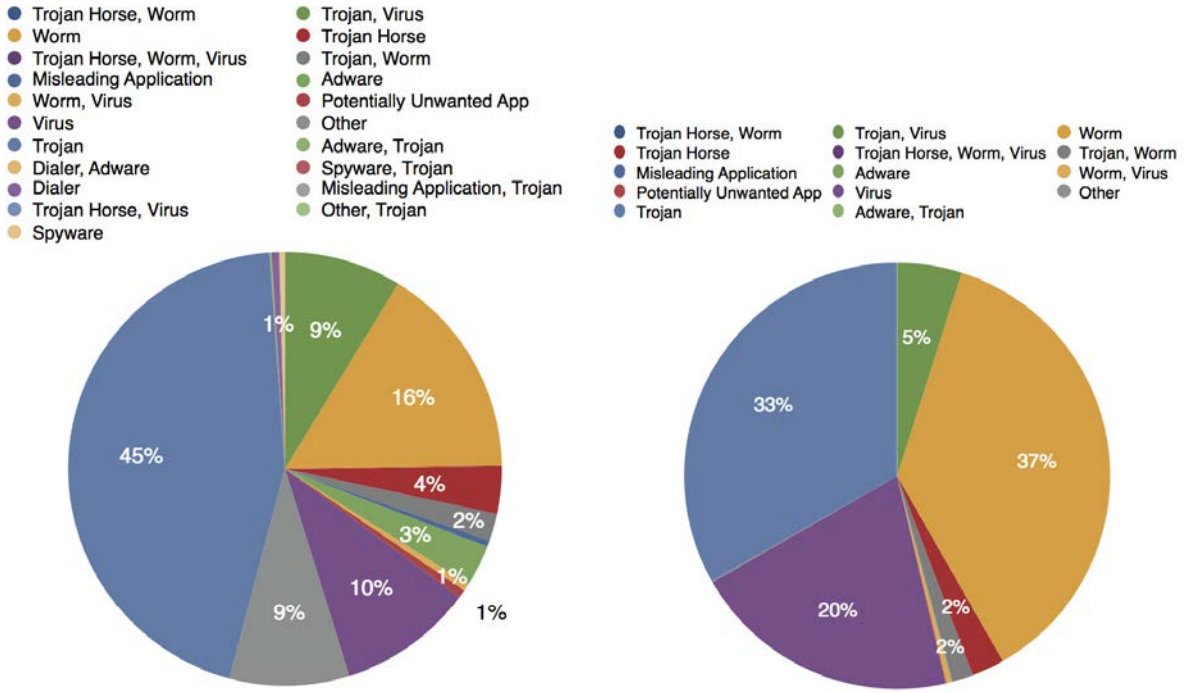


Figure 3.2: Malware samples categorised by their classes or types.



(a) Anubis

(b) Cuckoo

Figure 3.3: Malware classes analysed by Anubis and Cuckoo Sandbox.

Table 3.1: The top 10 malware variants analysed by each sandbox

Anubis			Cuckoo Sandbox		
Malware Variant	**#**	**Date**	**Malware Variant**	**#**	**Date**
Trojan.Gen	1067	2010/02/19	Trojan.Gen	202	2010/02/19
WS.Reputation.1	808	NA	W32.Pilleuz!gen6	138	2010/09/29
Trojan Horse	335	2004/02/19	Trojan.Gen.2	137	2010/08/20
Trojan.ADH	294	2010/03/10	W32.Spybot.Worm	136	2003/04/16
Backdoor.Trojan	280	1999/02/11	Trojan.Gen.3	99	2013/08/06
W32.Spybot.Worm	280	2003/04/16	Downloader	99	2001/08/08
SMG.Heur!cg1	276	2015/08/28	W32.Pilleuz!gen40	98	2013/08/22
Downloader	272	2001/08/08	W32.Almanahe.B!inf	96	2007/04/15
W32.Sality.AE	253	2008/04/20	W32.Sality.AF	93	2014/01/02
Trojan.ADH.2	223	2010/08/30	W32.Sality.Y!inf	91	2007/03/16

3.6 Malicious Behaviour Observed

This section provides an overview of some of the activities carried out by the malicious samples collected during the analysis process; examples of the activities covered are files and registry activities. The aim of this section is to provide an insight into malicious behaviour that is common across the different families and types of malware in our knoweldge-base, and to show how an observed activity, such as a change in the reg-

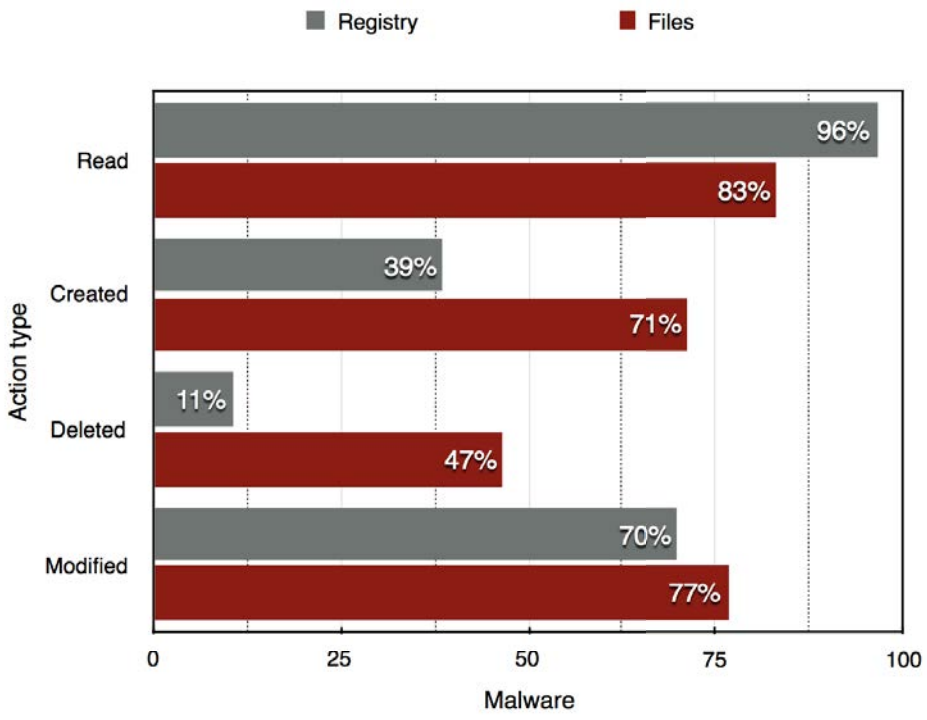


Figure 3.4: Activities carried out on the system's files and registries.

istry keys or values, could be used as an indicator of a possible malicious infection as will be seen later on, in Chapters 5, and 6.

Figure 3.4 shows the percentage of malware samples for each of the malicious activities carried out on the system's files and registry. The figure shows that 71% of malware samples created new files during the infection process; in particular, 436 samples created a file under the "`extensions`" folder, which belong to the Firefox browser; this can be for the purpose of monitoring users' browsing activities (e.g. stealing their sensitive information, such as passwords, financial information and others). Also 1173 samples created files under the "`\Temporary Internet Files\Content.IE5`" folder. This folder is a preferred place for malware to download their extra files in preparation for further malicious activities (e.g. attacking or exploiting vulnerabilities in the infected system). The reason for choosing such a folder to download the malicious files might be because it is hard for users to inspect it, due to the large number of cached files present in that location [199, 122].

Figure 3.4 shows a large percentage of the malware deleted files from the infected

system. By analysing most of these deleted files we found the majority were from the system "temporary" folders, such as "\Temporary Internet Files\Content.IE5", suggesting that the malware was trying to hide itself by deleting previously created files.

Table 3.2: Overview of the observed registry activities

ID	Registry paths	%
Reg_1	\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\ FirewallPolicy\StandardProfile\AuthorizedApplications\List\	4.21
Reg_2	\Software\Microsoft\Windows\CurrentVersion\Run\	27.21
Reg_3	\Software\Microsoft\Windows\CurrentVersion\policies\system\	10.51
Reg_4	\Software\Microsoft\SystemCertificates\TrustedPublisher\Certificates\	1.06

In terms of the registry activities, Figure 3.4 suggests that malware typically tends to access registry keys to read a value, while only a small percentage of malware (i.e. 11%) deleted a value during the infection. Malware can access a single registry path, such as '/SOFTWARE/Microsoft/Windows/CurrentVersion/policies/system' to carry out various malicious activities, e.g. modifying values like "DisableTaskMgr"=1, "DisableRegistryTools"=1 or "EnableLUA"=0 which indicates, respectively, disabling registry editing tools known as *regedit*, the Windows task manager, or the User Account Controls (UAC) responsible for notifying the user when a program is trying to make changes to the computer like installing software. We have checked whether malware accessed some interesting registry paths, which are listed on Table 3.2. As mentioned above, each of the listed paths can be accessed and modified in various ways to adopt different functionalities; malware can add a new value to Reg_1 to bypass the Windows firewall and connect to the internet without making the Windows firewall trigger any warning. While malware access a registry such as Reg_2 either, to infect executables that run at Windows startup, or to add the malware to the startup list. On the other hand, Reg_3 can be modified to disable the registry editing tools like regedit, the Windows task manager, or the User Account Controls (UAC), which is responsible for notifying the user when a program is trying to make some changes to the computer such as installing software. Also, malware can access Reg_4 for installing their own certificate as trusted.

During the analysis process, we have also seen a number of malware samples that tried

Listing 3.3: A snapshot of the report generated by Cuckoo Sandbox for W32.Ramnit.B!gen2 malware variant.

```
 "families": [],
"description": "Tries␣to␣unhook␣Windows␣functions␣monitored␣by␣Cuckoo
   ",
"severity": 3,
"marks": [
    {
        "call": {
            "category": "__notification__",
            "status": 1,
            "stacktrace": [],
            "api": "__anomaly__",
            "return_value": 0,
            "arguments": {
                "subcategory": "exception",
                "tid": 1084,
                "message": "Encountered␣1025␣exceptions,␣quitting.",
                "function_name": ""
            },
            "time": 1463475488.0,
            "tid": 1084,
            "flags": {}
        },
        "pid": 796,
        "type": "call",
        "cid": 1039
    }
],
"references": [],
"name": "antisandbox_unhook"
```

Table 3.3: Unique API calls made only by malware

API call	**# malware**	**API call**	**# malware**
NtCreateThread	38	CreateDirectoryExW	1
RtlCreateUserThread	35	CertOpenSystemStoreW	23
CryptGenKey	5	NtCreateProcess	1
DnsQuery_A	27	RtlDecompressBuffer	107
GetBestInterfaceEx	1	Thread32First	233
RtlCompressBuffer	1	InternetWriteFile	1
NtLoadDriver	14	NtSaveKey	11
GetKeyboardState	107	FindFirstFileExA	27
Thread32Next	234	NtMakeTemporaryObject	7
WSASendTo	6	NtDeleteFile	3
GetAddrInfoW	3		

to evade being analysed by unhooking Windows functions monitored by the used sandbox (Cuckoo Sandbox in this case); this caused the sandbox monitor to throw as many as 1025 exceptions. An example of such samples was `W32.Ramnit.B!gen2`; Listing 3.3 shows a snapshot of the report generated after analysing the malware sample.

3.6.1 API Call Statistics

Besides the registry and files activities presented in the previous section, we have also examined the use of API function calls, which have been used as an input for the machine learning-based detection system presented in Chapter 4. We found that a total of 286 API functions were called, where most of them were shared among the two classes, malware and benign. However, after analysing those APIs further, we noticed that some of them were called by one class only and not the other. In the case of benign samples, there was only one unique API that was not used by malware, i.e. `NtDeleteKey`, whereas there were 21 unique API functions called only by malware. Table 3.3 shows those APIs which were called only by malware, along with the number of malware samples that made each call.

3.7 Conclusion

In this chapter, we reported the process of collecting a large and representative malware corpus, which can provide a rich and varied set of features to be used for the purpose of malware detection. We described the process of downloading, analysing and storing the raw malware features. We used the information gathered to generate an intermediate feature representation to be used as an input for different malware detection approaches, which are presented in the subsequent chapters. The contents of this chapter forms an important source of information for this thesis, as all the works presented were possible, and have been evaluated using features from the knowledge-base that has been presented.

CHAPTER 4

ANALYSIS OF THE MISCLASSIFICATION OF MACHINE LEARNING-BASED MALWARE DETECTION SYSTEMS

4.1 Introduction and Motivation

Past research has shown that machine learning-based detection systems can detect new malware using the knowledge derived from training a classifier on previously discovered and labeled malware samples (e.g. [12, 56, 136, 68]). However, due to the fact that malware is evolving and its behaviour can change, as in the case of exploiting a new vulnerability [187], or an attempt of malware writers to evade detection, malware could remain undetected, and therefore, be classified incorrectly as benign.

In this chapter we investigate the reasons behind the misclassification of malware, so that a deeper understanding of the causes could be developed, and therefore, the appropriate mitigation method could be used. We tracked changes adopted by the misclassified malware instances, and we investigated whether there was a recognisable pattern across these misclassified samples. In summary, in this chapter we have investigated the following sub-questions:

- Does misclassification increase over a period of time?
- Does misclassification occur in malware at the level of families, where all instances

that belong to specific new malware families are misclassified?

- Alternatively, does misclassification occur at the level of variants, and it is not related to malware families?
- When misclassification does occur can we find the reason for it?

In order to answer these questions, we used 5,410 malware samples, from approximately 400 variants drawn from 200 malware families. We built a classifier based on our malware and benign samples. We then conducted multiple experiments: in the first experiment we ran the classifier on new malware grouped by their year of discovery. In the second experiment we grouped each new malware family into its available variants. We recorded the detection rate resulting from each group, in each experiment, in order to answer the research questions mentioned above. We used two different, widely known classification algorithms in the context of malware detection (as discussed later on in this chapter, p. 67): Support Vector Machines (SVM), and Decision Trees (DT) as a base for our classifier to ensure that the results produced were not dependent on a specific classification algorithm.

We proceed as follows: Section 4.2 gives a sketch of the approach followed to answer the questions above. The classifier design, along with the experiments' results are described in Sections 4.3 and 4.4, respectively. The analysis is then presented in Section 4.5,whereas Section 4.6 describes the related work and determines how the current work presented differs. Section 4.7 discusses the research outcomes and outlines the conclusion.

The contents of this chapter are based primarily on one of our contributions to the literature (Alruhaily et al. [7]).

4.2 Sketch of the Approach

This section describes the approach employed to fulfil the main objective of this chapter, which is checking the reasons behind the misclassification. The work presented in this

chapter can be divided into the following tasks:

1. Collecting and analysing malware (described previously in detail in Chapter 3).

2. Building a typical ML-based detection system (i.e. a classifier). This task includes:

   - Extracting and determining the features that will be used when classifying the data.
   - Assessing the performance of the implemented classifier.

   The feature extraction step is described in Subsection 4.3.1, while the system architecture is illustrated in Subsection 4.3.3.

3. Classifying large grouped datasets. Following the previous task, we classified large datasets which have been grouped into:

   - Years: in order to check whether there is a notable change in malware behaviour over time, which might result in a change in the classification rate in one of the years. We chose one year intervals based on the amount of data we have, as it was the minimum period which was able to produce a stable results.
   - Variants: in order to check whether these changes, which affect the detection rate, could be traced back to specific malware families where all their variants are hard to detect, or just to particular variants, without the family membership being a factor.

   The details of these two experiments are described in detail in Section 4.4.

4. Analysing the misclassified instances, which includes analysis of the misclassification that occurred, and identification of its reasons. The analysis is discussed in Section 4.5.

4.3 Building a Classifier

In this section, we discuss the process of extracting the features that were used in addition to the classification procedure. This section also includes the metrics that were used to measure the performance throughout this chapter.

4.3.1 Feature Extraction

The different features' types used for malware have been described in detail in Chapter 3 p. 48. We used the API calls as our dynamic features as they have been widely used as behavioural features for such systems, and they also show that they can offer an accurate representation of malware behaviour [53, 54, 176, 136, 68]. Additionally, it is also worth noting that relying on behavioural analysis helps to avoid problems when dealing with some cases (such as malware obfuscation). Based on some preliminary tests, we found that the term frequency weight, (described earlier in Chapter 3, p. 53), of the APIs did not improve the classification any further, which is similar to the conclusion reached by [176], where only information related to the presence of an API is important and not the frequency. Thus we used the binary vector representation of the extracted APIs where '0' refers to the absence of an API call and '1' denotes its presence.

In order to transform the feature from the textual nature to vectors to be used as an input for a machine learning algorithm, we adopted a bag of n-gram representation (described in Chapter 3 p. 52) to represent the API call sequence, as it is widely used as a means for detecting malware and it shows promising results [5, 54], where unigram, bigram, a combination of unigram and bigram, and trigram have been evaluated. Using unigrams means that the sequence of API words is unimportant, where we simply check whether a word is present or not. By using bigram and trigram, the feature vector will not only contain a single API word; it will preserve the sequence of two and three words, respectively, which can be considered as taking a snapshot of the malware behaviour. We have documented the classifier performance when using each of the n-gram representations

in Subsection 4.3.3, where the classifier's design is described, along with the process of assessing its performance.

We also believe that using a hybrid analysis (both static and dynamic features) can boost the detection rate further. However, as we intend to use the classifier in checking malware behaviour in different scenarios: over time and with malware grouped into variants; static analysis is beyond the scope of this chapter.

4.3.2 Evaluation Metrics

To evaluate the performance of the classifier we have used a confusion matrix (described in Chapter 2 p. 25). Metrics derived from the confusion matrix were used to define four further performance metrics, which have been used throughout this chapter:

1. Sensitivity (recall): measures the proportion of true positives: Sensitivity = $\frac{TP}{TP+FN}$

2. Specificity: measures the proportion of true negatives: Specificity =$\frac{TN}{TN+FP}$

3. The Geometric mean (G-mean): also known as the macro-averaged accuracy [55], is the geometric mean of recall over all classes. It is recognised in the field of machine learning as a more accurate measure of performance than the normal accuracy in an unbalanced classification scenario, as it considers the accuracy of both classes: the majority and the minority [99][85]. The G-mean can be calculated as follows: G-mean=$\sqrt{Sensitivity \cdot Specificity}$

4. The area under the receiver operating characteristic curve (i.e. the area under the ROC curve, or for short AUC_{ROC}) [77]: The ROC curve is a graphical plot that illustrates the performance of a binary classifier. The curve is created by plotting the true positive rate (sensitivity) against the false positive rate (1-specificity) at various thresholds. The value of AUC_{ROC} can vary between 1.0 and 0, where 1.0 indicates a perfect classifier with an ideal separation of the two classes, and an AUC_{ROC} of 0.5 represents worthless classifier. AUC_{ROC} is insensitive to class imbalance; if

the majority labels of the data are positive or negative, a classifier which always outputs 1 or 0, respectively, will have a 0.5 score although it will achieve a very high accuracy. We calculated the AUC_{ROC} score based on the functions provided by the *Scikit-learn* library, where the library offers a wide range of machine learning algorithms and tools [149].

Both metrics, G-mean and AUC_{ROC}, are commonly used to evaluate the imbalanced data classification performance. However as AUC_{ROC} considers all the possible thresholds, it is commonly used to assess the performance when choosing the best model that will be used during the classification procedure. Therefore we will be using both G-mean and AUC_{ROC} to assess and choose the best model in Section 4.3.3, while we will be using the G-mean, which represents the balanced accuracy, as our main metric when classifying malware based on their year of discovery in Section 4.4.1. In Section 4.4.2 we will be using sensitivity as our main metric as it conforms with the goal of our second experiment, which focused on the classifier's ability to correctly identify malicious instances without adding any noisy data related to the benign samples.

4.3.3 Classifier Design

For our classifiers, we used Support Vector Machine and Decision Tree algorithms as base classifiers due to the fact that they are widely known machine learning algorithms [40, 91]. In addition, they have shown State-of-the-Art results in a number of classification problems, including classifying and recognising new malware [206, 207, 204, 101, 5, 93]. SVM and DT classifiers were implemented based on the linear kernel function, and C4.5 algorithm, respectively. The classifiers were also used with their default parameters as provided by the *Scikit-learn* library (version 0.17.1).

As mentioned previously, we have also evaluated their performance with different sizes of n-grams: unigram, bigram, unigram+bigram and trigram, as shown in Table 4.1, and we have chosen the best settings for each classifier. In the case of SVM, the bigram and

the unigram+bigram both gave us the best results; however, as our aim is to test malware discovered in the followed years where a match of an exact sequence might not be found, we thus preferred to use the unigram+bigram, which will result in more generalisation as it tests the occurrence of a single API word, in addition to a sequence of two APIs together.

Table 4.1: Classifiers' performance on different n-gram sizes.

	SVM		DT	
Feature set	G-mean	AUC_{ROC}	G-mean	AUC_{ROC}
API_{Uni}	0.92519	0.97163	0.92796	0.97203
API_{Bigram}	0.93965	0.97681	0.91791	0.96815
$API_{Uni+Bigram}$	0.94041	0.97692	0.91965	0.96865
API_{Tigram}	0.92465	0.97079	0.89926	0.95985

We have an imbalanced dataset; where the malware class represents the majority class. The class imbalance is a common problem in the area of machine learning in general [202], and in malware detection in particular [116, 205]. Such a problem occurs when the number of instances in each class varies (as described in Chapter 2, p. 24). In malware detection systems, the imbalance problem is due to the fact that malware can be downloaded in large numbers from open access database such as OpenMalware [127], VXHeaven [186], VirusShare [183], whereas it is more difficult to gather benign samples [107]. The imbalance problem can affect the classification process as the classifier becomes more biased towards the majority class. To avoid the effect of this problem, we tested a well-known approach in the area of machine learning that is referred to as *Exactly Balanced Bagging* [34, 85, 87], (described in Chapter 2, p. 24). This approach is based on classifying balanced subsets; it is a modified version of *Bagging* [27], a method that has been used extensively in malware detection systems with different base classifiers and which gives a considerably higher detection rate than the normal classifiers [204], even with an imbalanced dataset [134]. To the best of our knowledge, this work is the first that has used EBBag for malware detection.

Bagging (as described in Section 2.3.1), is a classifier ensemble technique that is based

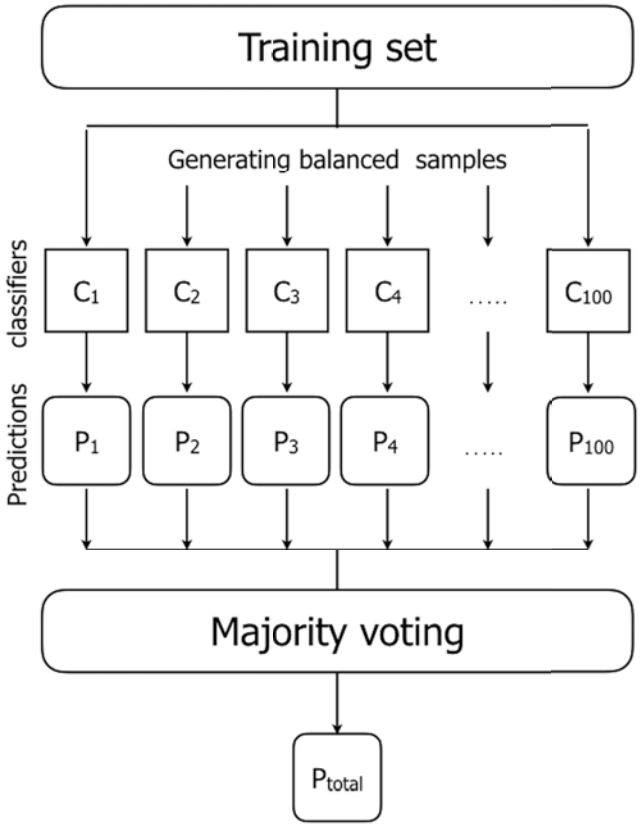


Figure 4.1: The EBBag model

on randomly drawn subsets of the training data. Each time a subset is drawn, a classifier will be constructed to classify the newly generated subsets. The classification procedure is repeated a number of times (we used 100 times, as in [204]) and majority voting over all predictions is calculated as the final prediction to ensure the robustness of the results.

Our framework implemented the EBBag approach, which is based on Bagging, but with a minor modification to the way the subsets are drawn. In EBBag, the entire minority class is used for each classifier, along with randomly generated subsets of the majority class, which are the same size as the minority class, so balancing the data. The procedure of generating smaller samples is known as "downsampling". Figure 4.1 depicts the classifier model.

We ran five tests to compare Bagging with EBBag, calculating the classification rate of randomly chosen malware and benign samples, with a malware to benign ratio varying between 2:1 and 10:1. We performed 10-fold cross validation and compared our adopted

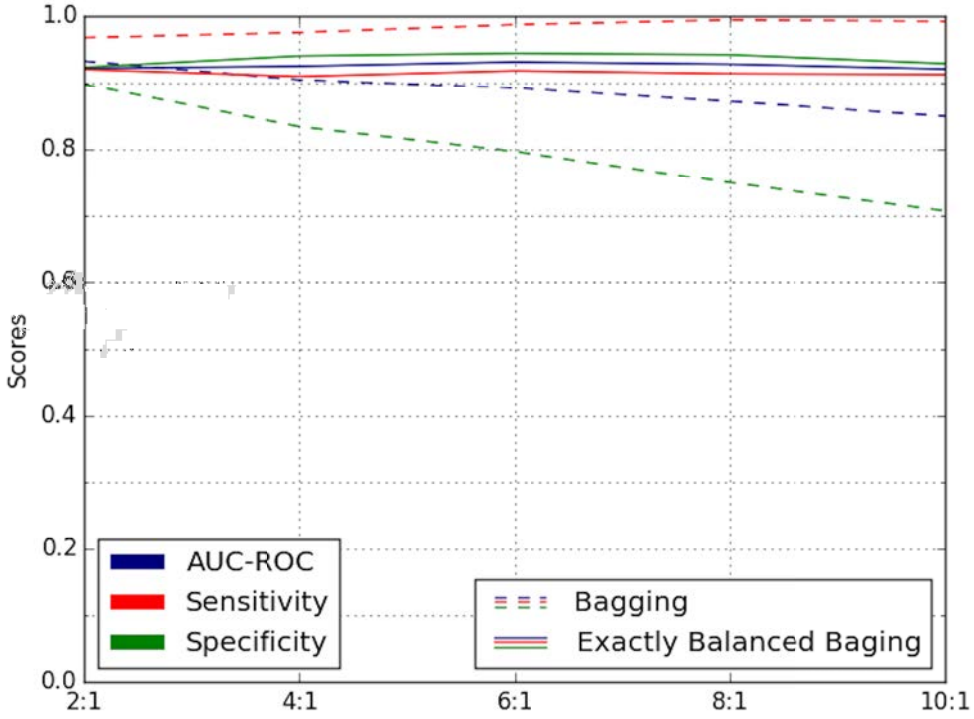


Figure 4.2: Classification rate with different ratio of malware samples to benign.

approach, EBBag, to the Bagging approach with the same base classifier (SVM) and the best n-gram size. The results of these tests are shown in Figure 4.2. The Figure shows the true positive rate (sensitivity), the true negative rate (specificity) and the AUC_{ROC} recorded for each approach. We note that Bagging becomes increasingly inaccurate as the data becomes more imbalanced. So the figure indicates that imbalanced data will be part of the cause of the misclassification rate in papers that use Bagging alone to classify malware.

By using EBBag, it can be seen that the measures: sensitivity and specificity, which represent the accuracy of the malicious and benign classes, respectively, have not been affected by the imbalance problem. Also, the false positive rate (1-specificity) is significantly decreased from 0.29 to 0.07, where a false positive occurs when a classifier flags a benign file as malicious by mistake; this is usually costly as it consumes a considerable amount of resources and time. Therefore, this analysis shows that EBBag outperforms Bagging when dealing with an imbalanced dataset; thus we use EBBag for the rest of the work.

4.4 Classifying a Large Grouped Dataset

As mentioned previously, in order to answer the main research questions of this chapter we have followed two methods when testing the data:

1. Classifying all malware based on the year of discovery.

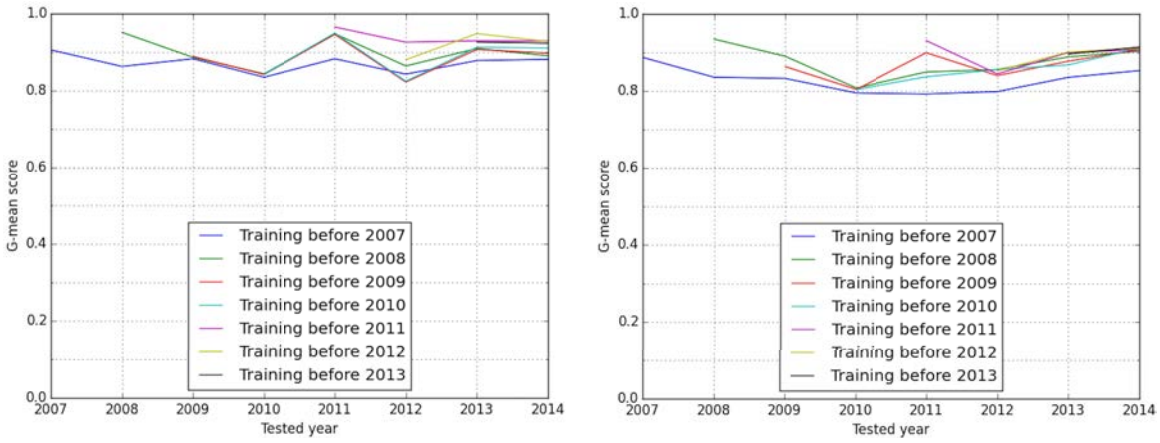
2. Classifying malware based on malware variants.

Sections 4.4.1 and 4.4.2 describe in detail the process used in these two methods.

4.4.1 Classifying Malware Based on the Year of Discovery

In our first experiment, we tested the classifier using data grouped by year in order to check whether there was a notable change in malware behaviour over time; this will be represented as a change in the classification rate (as described in Subsection 2.3.4). We divided the entire test set into years from 2007 to 2014, based on the discovery date recorded by Symantec; accordingly, we ended up with 8 testing sets. We chose one year intervals based on the amount of data we had, as it was the minimum period which was able to produce stable results; i.e. with a minimum fluctuation in the classification rate, where the fluctuation was mainly due to the lack of malware samples for smaller intervals (e.g. months or weeks). The initial training set included samples until 2006. Each time, the training set was evaluated separately on each of the years followed; so for the initial training set it was evaluated on malware discovered from 2007 to 2014. We repeated the experiment by extending the training set to include the year followed and testing on the remaining years.

Figure 4.3 shows the averaged accuracy (G-mean) recorded by our framework on data trained on real malware samples. Most of the results showed that the classifier can maintain a score above 80%, which is comparable to other works in the area of malware detection, especially in studies whose focus was to explore the effect of time on the malware classification rate (e.g. [81]). It can also be seen that the more the data included



(a) Using SVM (b) Using DT

Figure 4.3: Malware tested yearly.

in the training phase, the higher the G-mean score for the following year will be, except for some cases where a new family was introduced (such as in 2009 and 2010), or where a new variant with a slightly different behaviour was introduced such as in 2012 with `W32.Ramnit.B!gen2`. In the case of this family, although some variants of the malware family `W32.Ramnit` were introduced before 2012 and the system was trained on some of them, it seems that the `W32.Ramnit.B!gen2` variant which was introduced in 2012 implements some anti-sandboxing techniques, where the malware tried to unhook Windows functions monitored by the sandbox, causing the Cuckoo monitor to throw 1025 exceptions. A snapshot of the report generated for this variant is shown in Chapter 3 p. 58.

However, from Figure 4.3 we can conclude that the detection rate is not consistently affected by the passage of time; instead, the classifiers, generally, can continue providing a good detection rate. Thus, to analyse the results further, we carried out another experiment which is explained in the next section. The experiment aimed to check whether the misclassification caused by the changes in malware behaviour can be traced back to a number of malware families or even to sub-families (a.k.a variants).

4.4.2 Classifying Malware Based on Malware Variants

In this experiment we tested seven malware families (broken down into variants) in order to check whether the misclassified malware instances were a result of undetected behavioural changes at the malware family level, or whether they were caused by other changes in the level of variants.

The experiment was conducted for seven families, namely: `W32.Changeup`, `W32.Pilleuz`, `W32.Imaut`, `W32.Sality`, `Trojan.FakeAV`, `Trojan.Zbot` and `Trojan.Gen`. The remaining families were not used due to an insufficient amount of samples for each particular variant in the available data set. These seven families correspond to a total of 2737 malware samples. For testing each of which families, we trained the classifier on malware data prior to 2007. This is because we would have after this date a reasonable number of malware families grouped into variants to carry out the testing process.

We used sensitivity here as our main metric in the experiment (referred to in the table as *Sens*) to measure the detection rate as it conforms with our goal, which is focusing on the classifiers' ability to correctly identify those malicious instances without adding any noisy data related to the benign samples, unlike the G-mean metric, (i.e. used in the previous experiment), which provides the averaged accuracy of the two classes.

Table 4.2 shows the results of classifying each of the malware families, including their available variants. It shows that most of the behavioural changes, which caused the misclassification of malware, could be traced back to a number of variants without the family membership being a factor; meaning that in most cases we would not expect the changes to be replicated across variants of the same malware family. This implies that we might see a single variant that was hard to detect in a family of malware in which the other variants could be reliably detected. Therefore, due to the fact that there is no absolute pattern (as seen in Table 4.2) between the misclassification rate and the malware family, nor with the discovered year of each of the variants, it could be concluded that, the misclassification could be mostly linked to several malware variants and it cannot be traced back to the discovery date of each of the tested variants, nor to changes at the

Table 4.2: Malware tested by classifiers trained before 2007

	Variants	#	Date	SVM-Sens	DT-Sens
Changeup	W32.Changeup	41	18/08/09	0.98	0.98
	W32.Changeup!gen9	48	02/09/10	1.0	1.0
	W32.Changeup!gen10	47	02/02/11	1.0	1.0
	W32.Changeup!gen12	47	11/08/11	1.0	1.0
	W32.Changeup!gen16	51	20/07/12	1.0	1.0
	W32.Changeup!gen44	76	01/08/13	0.99	1.0
	W32.Changeup!gen46	67	28/05/14	1.0	1.0
	W32.Changeup!gen49	64	20/08/14	0.94	1.0
Pilleuz	W32.Pilleuz!gen1	47	19/01/10	0.77	0.68
	W32.Pilleuz!gen6	138	29/09/10	0.88	0.86
	W32.Pilleuz!gen19	70	17/01/11	0.97	0.79
	W32.Pilleuz!gen21	64	29/03/11	0.28	0.03
	W32.Pilleuz!gen30	46	01/02/12	1.0	1.0
	W32.Pilleuz!gen36	60	07/02/13	1.0	1.0
	W32.Pilleuz!gen40	98	22/08/13	1.0	1.0
Imaut	W32.Imaut.AA	68	07/06/07	0.97	0.80
	W32.Imaut.AS	45	01/08/07	0.84	0.84
	W32.Imaut.CN	73	20/02/08	0.85	0.92
	W32.Imaut.E	64	23/12/08	0.88	0.83
	W32.Imaut!gen1	46	20/09/10	0.24	0.04
Sality	W32.Sality.X	46	12/01/07	0.96	0.93
	W32.Sality.Y!inf	91	16/03/07	0.98	0.98
	W32.Sality.AB	55	11/01/08	0.02	0.02
	W32.Sality.AE	71	20/04/08	0.93	0.87
	W32.Sality.AM	51	18/04/09	0.80	0.75
	W32.Sality!dr	71	31/08/10	0.80	0.80
	W32.Sality!dam	54	30/04/13	0.15	0.15
	W32.Sality.AF	93	02/01/14	0.90	0.77
FakeAV	Trojan.FakeAV	41	10/10/07	0.68	0.85
	Trojan.FakeAV!gen29	70	07/05/10	0.99	0.93
	Trojan.FakeAV!gen99	38	08/03/13	1.0	1.0
	Trojan.FakeAV!gen119	42	01/04/14	0.29	0.12
Zbot	Trojan.Zbot	40	10/01/10	0.98	0.28
	Trojan.Zbot!gen9	48	16/08/10	1.0	0.94
	Trojan.Zbot!gen43	48	26/05/13	0.85	0.88
	Trojan.Zbot!gen71	44	23/12/13	1.0	0.11
	Trojan.Zbot!gen75	32	05/06/14	0.69	0.97
Gen	Trojan.Gen	202	19/02/10	0.48	0.77
	Trojan.Gen.2	137	20/08/10	0.45	0.45
	Trojan.Gen.X	52	12/01/12	0.42	0.42
	Trojan.Gen.SMH	52	26/10/12	0.62	0.40
	Trojan.Gen.3	99	06/08/13	0.61	0.60

level of the malware family (which would affect all of that familie's future variants).

4.5 Reasons for Misclassification

The aim in this section is to analyse the classification results. This includes also outlining the differences between the correctly classified and the misclassified variants and explaining the reasons that may lead to the misclassification. Generally, from Table 4.2 we can identify three misclassification cases, although most of the misclassifications occurred at the level of variants. We can summarise the different misclassification cases as follows:

- Variants misclassified by both classifiers.
- Variants misclassified by only one classifier.
- Misclassification which occurred at the family level instead of variants.

4.5.1 Variants Misclassified by both Classifiers

Table 4.2 shows a case where more than 70% of malware samples that belong to specific variants were misclassified by both classifiers: SVM and DT. These variants are: W32.Pilleuze!gen21, W32.Sality.AB, W32.Sality!dam, W32.Imaut!gen1 and Trojan.FakeAV!gen119.

In the case of the W32.Pilleuze!gen21, W32.Sality!dam and W32.Sality.AB variants, it seems that these variants had not performed any behavioural action when being analysed. This can happen because the samples implemented some anti-virtualisation techniques, or because they were looking for a specific argument, or because they were corrupted files. All three variants mentioned terminated the process by calling NtTerminateProcess. In the case of W32.Sality.AB and W32.Sality!dam they also adopted a stealthiness technique, where they disabled the error messages through calling SetErrorMode API with the arguments SEM_NOGPFAULTERRORBOX — SEM_ NOOPENFILEERRORBOX. After looking at the W32.Sality!dam page on Symantec [165], it

seems that this variant is considered as a corrupted file where it can no longer be executed or infect other files. While removing all examples of misclassified corrupted malware from our dataset would have been possible, we note no other work on malware classification does this. Therefore, removing these samples would not reflect other work. The W32.Imaut!gen1 worm, on the other hand, did not terminate the process; however, it did not perform any network activity which might lead to its being misclassified. In fact, only 2 samples out of the 46 carried out some network activities and both of them had been classified correctly.

In the case of Trojan.FakeAV!gen119, the malware variant used an uncommon API, (compared to others in our database), to connect to the Internet: InternetOpenW, InternetOpenUrlW, which are the Unicode based API of the high-level Internet API: Windows Internet (WinINet). The calls used by this variant take arguments in Unicode format, while the older variants of this malware family used the ASCII based API calls instead: InternetOpenA, InternetOpenUrlA. This raises the question: would normalising the API by removing the appended characters such as A, W, ExW, and ExA when processing the features, such as having only InternetOpen in the features set instead of multiple entries increase the overall accuracy of API based classifiers?

To answer this question, we carried out another experiment by using all of our data and normalised the Win32 API by removing the appended characters, such as A, W, ExA and ExW. We then performed 10-fold cross validation to assess the performance of the classifier, using the stemmed and un-stemmed features set. The results are shown in Table 4.3. It can be seen from the Table that removing the appended letters did not have a considerable impact on the classification rate. However, by removing the appended characters we ended up with around 230 features instead of 280, without significantly affecting the detection rate. Such an option can be considered to improve the efficiency of the classifier and to minimise the time needed for the classification. We note that other papers [144, 181, 143] use feature selection methods to reduce the number of features in their datasets; however, they do not use API stemming, which Table 4.3 suggests might

be a helpful addition.

It is important to note that conclusions drawn in this section were based on the assumption that the increase in the misclassification rate were mainly due to changes in malware behaviour. However, there is another possibility that has not been considered here, which is the reasonability of the features used by the classifier to predict the true classes (i.e. malware or benign), and thus the effect of the choice of the classifier on the misclassification rate. This aspect has been left for future work, as described later on in Chapter 7, p. 124, where some advanced tools such as LIME [139], could be utilised to explain the predictions of black box text classification algorithms such as SVMs.

Table 4.3: Performance on stemmed and un-stemmed features set.

	un-stemmed 286 API		stemmed ≋ 230 API	
	G-mean	AUC	G-mean	AUC
SVM	0.94747	0.94747	0.94681	0.94682
DT	0.92381	0.92393	0.91688	0.91701

4.5.2 Variants Misclassified by Only One Classifier

Another case which we have investigated can be seen in `Trojan.Zbot` and `Trojan.Zbot!gen71`, where nearly all their instances have been correctly classified by the SVM classifier; however, DT failed to classify as many as 89% of the samples correctly. We analysed random trees constructed by the classifier in order to be able to determine the reasons behind the misclassification (Figure 4.4 shows an example of a simplified version of a tree constructed by DT classifier). In the case of the `Trojan.Zbot`, it seems that the absence of the call: `SetWindowsHookExA` was the reason for the misclassification on almost all the variant's samples. While in the case of the `Trojan.Zbot!gen71` variant, the correctly classified malware called the `NtProtectVirtualMemory` API to allocate a read-write-execute memory. This API is usually called by malware binaries during the unpacking process, and the absence of this call might indicate that the malware sample

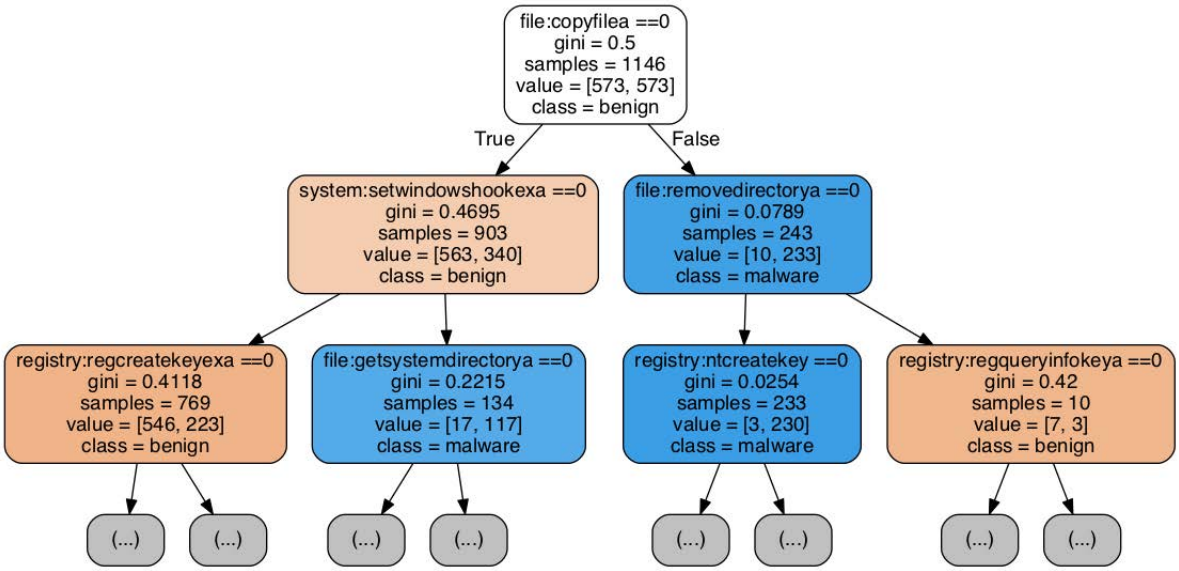


Figure 4.4: A simplified version of the tree constructed by DT algorithm based on one run of the EBBag classifier.

was not packed, which, in our case, led to the misclassification of the incorrectly classified instances.

The analysis results of the misclassification occurred in both cases: when using SVM classifier alone, or when using both classifiers, could give researchers valuable information when interpreting the successful rate produced by a machine learning-based detection system. The results also demonstrate the necessity of utilising a cleaning phase prior to the classification of malware samples, as the effectiveness of the classification algorithm could be greatly affected by the features fed into the classifier during the learning process [19].

4.5.3 Misclassification at Malware Family Level

Although most of the misclassification that we have seen occurred at the variant level, there is a single case where the misclassification can be linked to the family instead, as can be seen in Table 4.2 in the case of the `Trojan.Gen` family, where the SVM's and DT's classification rate of all the available variants ranged between 40% and 77%. The reason that this family is different from the others seems to be due to the fact that this family is actually a category of varied Trojans whose definitions have not yet been

formulated, as mentioned by Symantec on the `Trojan.Gen` family page [167]. By checking the misclassified samples and the paths that were taken by samples belonging to this family, it could be seen that although the samples may share some general characteristics, they adopted different techniques and thus the samples could behave in various ways and take different paths on trees generated by DT classifier (approximately 15 different behavioural paths), unlike other families where their behaviour was very uniform (2 or 3 paths).

As we have said, the behavioural profiles and definitions that resulted from this family were varied and thus we are only giving examples for some of the misclassification cases, as identifying all the reasons for the misclassification for this family would not be possible. Many of the misclassified instances did not connect to the internet either, because the malware were applying some anti-virtualisation techniques; an example of this case is: `Trojan.Gen.X`; or they were terminating the process as a specific argument had not been found, as in `Trojan.Gen.3`. In the case of `Trojan.Gen.X`, nearly half of the misclassified samples belonging to this variant were monitoring the user window by calling `GetForegroundWindow` and checking the mouse movement by calling the `GetCursorPos` API. They also followed these calls by calling `GetKeyState` to monitor the following keys constantly: the mouse keys, Alt, Ctrl, and shift key. The execution was then delayed by going on a loop when monitoring these actions and `NtDelayExecution` has also been called. These techniques have been noticed when analysing recent malware, as reported by malware researchers in order to evade sandbox detection [147], and this could be the reason why all the variants that used that technique were misclassified.

4.6 Comparison with the Related Work

In the following section we review the literature on malware classification and detection systems that is directly related to our work, and we indicate how this work differs.

As seen in Chapter 2, most of the proposed ML-based malware detection systems have

been tested on either a limited or a current set of malware. This poses the need to examine the effect of the passage of time on such malware detection systems and explore whether this could affect the systems' detection rate. Motivated by this, Islam et al. in [80] showed that it is possible for current ML-based malware detection systems to maintain a high detection rate even when classifying new 'un-seen' malware samples. They considered two sets of malware in order to verify this idea, one collected between 2002 and 2007, and the other collected from 2009 to 2010. In their experiment they dated malware based on the collection date and they used static, behavioural, and a combination of both features during their experiment.

Singh et al. [156] used three approaches to detect changes in the distribution of features of malware, which can be a result of trying to evade the detection. They used relative similarity, meta-features, and retraining a classifier. They focused in their experiments on malware static features only, and on three malware families. Although the size of their malware sample was limited, their work has also provided evidence in favour of negligible change in regards to the tested features.

Shabtai et al. [151] on the other hand, used Opcode n-grams pattern as features. They addressed a question concerning the time span within which a classifier would be able to maintain an acceptably high detection rate, and when it should be updated with new malware instances (i.e retrained). Their research showed that classifiers can maintain a reliable level of accuracy. However, when testing malware released in 2007, a significant decrease in the accuracy was observed. They have stated that this could indicate that new types of malware were released during that year.

In this chapter we did not intend to propose a new machine learning-based malware detection system; instead, our aim was to develop a deeper understanding of malware behaviour and the misclassification rate resulting from using such detection systems. In summary, our work on this chapter is different from the research mentioned above in the following respects:

- In addition to looking at malware behaviour over time, we also tracked the misclas-

sification and investigated whether it can be traced back to malware families, or even to variants.

- We also investigated the possible reasons that led to this misclassification and analysed the results.

4.7 Conclusion

In this chapter, we classified malware grouped by their year of discovery, in addition to grouping them into malware variants. We tracked the misclassified malware instances and we investigated whether there were recognisable patterns across these misclassified samples. From our first experiment we found that classifiers can continue to give a high detection rate even after a period of time, which means that there is no correlation between the passage of time and the misclassification that occurred, despite a minor rise of the detection rate on the following year. We then concluded from our second experiment that, mostly, there were few recognisable patterns between the misclassification and malware families, as with the discovered year of malware variants. Instead, most of the misclassifications can be traced back to several malware variants. This variation, which occurred on the variant level, is due to the fact that some variants apply some anti-sandboxing techniques, or due to that some samples were looking for a specific argument to run, or due to the fact that some variants are actually considered as bad data. This conclusion can help in interpreting the successful rate achieved when proposing a machine learning-based detection system. It also demonstrates the need for carrying out a cleaning process before transforming the features and before using them to train a classifier, as this can affect the effectiveness of the proposed detection system by adding some noise to the generated features; a fact that is usually not considered when proposing and assessing the performance of a new malware detection system.

The situation where only some variants were unrecognisable might be due to the fact that malware writers are not yet trying to evade ML techniques as they are still considered

as new techniques to detect malware, and have not been used as widely as the signature-based ones. If these recognition systems became more common, it might be the case that we would see more of these techniques to evade the detection by such systems.

CHAPTER 5

A PROBABILISTIC APPROACH FOR EFFICIENT DETECTION OF MALWARE IN THE CLOUD VIA FORENSIC VIRTUAL MACHINES

5.1 Introduction and Motivation

Behaviour-based malware detection systems are heavily dependent on features extracted and collected from the systems examined (as seen in the previous chapter). It is thus of the utmost importance that the information gathered is comprehensive, and trustworthy; therefore, the infection status of the target system could be correctly identified. In the case of cloud computing, a number of Virtual Machine Introspection-based malware monitoring approaches (described in detail in Chapter 2, p. 31) have been proposed for inspecting the cloud's VMs in order to provide such information. These approaches are characterised by being isolated from the infected system, making them hard to be tampered with (i.e. trustworthy); and by retaining full visibility over the guest's activities (i.e. comprehensive). In addition to these advantages, such a VMI-based approach helps in reducing the amount of VM resources used during the scan, as the monitoring process is performed externally. The amount of resources used could also be reduced further by subdividing the monitoring task between small scanners, where each scanner identifies a single symptom, these small scanners are referred to in the literature as Forensic Virtual Machines.

Forensic Virtual Machines (as described in Chapter 2, p. 34), which is an architecture for using VMI technology, benefits from mini-VMs that perform a distributed monitoring to detect symptoms of malicious behaviour from outside the target VM. To reduce the costly use of resources; a single FVM is used for checking multiple VMs, rather than creating a dedicated FVM per VM. To arrange an FVM movement from one VM to another, it is initialised with one of many pre-programmed strategies (i.e. mobility algorithms). Mobility algorithms can vary from simple to dynamic algorithms such as scanning according to a random movement, a pre-determined order, or based on more advanced scanning algorithms. Although multiple algorithms have been introduced (e.g. in [69, 9]), there is no work that provides a comparison of these algorithms. Identifying the most beneficial mobility algorithm prior to the initialisation of FVMs is beneficial, as the creation of FVMs is known to be a resource-intensive task [23].

This chapter presents a probabilistic approach that helps in identifying the most beneficial mobility algorithm given parameters that represent the scanned environment, such as the number of VMs, the deployed FVMs and the cost of false positives and false negatives. The evaluation of the proposed approach showed that there was a considerable difference in the scanning cost when using different mobility algorithm; therefore, the best algorithm could be identified based on a comparison of the resulting cost of the scan (in terms of the scanning resources used, and the accurate identification of infections).

We proceed as follows: Section 5.2 describes the symptoms used during the scan. The main probabilistic approach is described in Section 5.3, while the framework that was developed based on the approach and used to evaluate it is described in Section 5.4. The results of the experiment along with the evaluation are then proposed in Section 5.5. Section 5.6 reviews the related work, and describes how the contribution presented differs. The chapter is then concluded with Section 5.7.

The contents of this chapter are based primarily on one of our contributions to the research literature (Alruhaily et al. [6]).

Table 5.1: The configurations examined

Family	S	Registry paths
Popuppers	S_1	/Volatile Environment/*
	S_2	/Software/Microsoft/windows/CurrentVersion/Internet Settings/Connections/*
	S_3	Software/Microsoft/Windows/CurrentVersion/Internet Settings/Zones/*
Rontokbro	S_4	/Software/Microsoft/Windows/CurrentVersion/Explorer/Advanced/*
	S_5	/Software/Microsoft/Windows/CurrentVersion/Policies/Explorer/run/*
	S_6	/System/CurrentControlSet/Control/SafeBoot/*
Sality	S_7	/CurrentControlSet/Services/SharedAccess/Parameters/FirewallPolicy/StandardProfile/AuthorizedApplications/List/*
	S_8	/Software/Microsoft/Windows/CurrentVersion/Run/*
	S_9	/Software/Microsoft/Windows/CurrentVersion/policies/system/*
FakeAV	S_{10}	/Software/Policies/Microsoft/Windows/Safer/CodeIdentifiers/*
	S_{11}	/System/CurrentControlSet/Control/Session Manager/*
	S_{12}	/Software/Microsoft/Windows/CurrentVersion/Internet Settings/*

5.2 Malicious Symptoms

In order to fully understand and follow the proposed probabilistic approach presented in the following sections of this chapter, we describe in this section those symptoms which will be examined during the scanning process by the set of FVMs. In the context of FVMs, a symptom refers to a characteristic that might arise as a part of malicious behaviour, such that its existence could indicate a malicious infection [69]. Examples of symptoms have been shown earlier in Chapter 3, p. 55; symptoms could include missing processes (e.g. Antivirus agents)[76], modifying registry keys or values, and changing the file attributes (e.g. the creation time)[21].

Registry key manipulations have been used here as symptoms, because they offer a wealth of information as sources of forensic evidence [31, 105]. We grouped the set of symptoms into configurations (as in [69]), where $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ denotes the set of all important configurations. Each configuration $c \in \mathcal{C}$ consists of a set of symptoms that is used to detect a specific malware infection (a malware family or type); examples of such sets of symptoms are presented in Table 5.1. The table highlights common registries manipulated by four malware families, namely: `Adware.Popuppers`, `W32.Rontokbro@mm`,

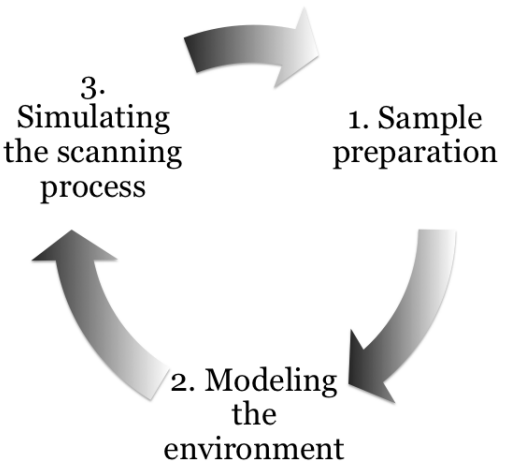


Figure 5.1: A high-level view of the proposed approach.

`W32.Sality` and `Trojan.FakeAV`. Each registry path S_i could be accessed and modified in various ways, allowing a wide range of functionalities.

Taking registries accessed by the Sality malware family as an example, malware could add a new value to S_7 to bypass the Windows firewall, connecting to the internet and preventing the Windows firewall from triggering a warning. While it uses registries such as S_8 to infect executables that run at Windows startup, it might also add the malware to the startup list. S_9 can be modified to disable registry editing tools like *regedit*, the Windows task manager or the User Account Controls (UAC), which are responsible for notifying the user when a program tries to make changes to the computer, as when installing software. For some of the previous symptoms such as S_8, the occurrence of the symptom might not always mean that a machine is infected, as some benign applications tend to access the same registry path and modify related registry values to add themselves to the startup list. The proposed approach, described in detail in Section 5.3, is based on Bayes' theorem; therefore, by using a configuration such as one of those presented in Table 5.1, it could be asserted that a Virtual Machine is infected, based on, the probability of it being compromised, given an inspected symptom (S_i), or a set of symptoms ($\mathcal{S}$) observed concurrently.

5.3 The Probabilistic Approach

The main problem that this chapter addresses is identifying the optimum mobility algorithm, which could be used in the scanning process, prior to the initialisation of FVMs. We have benefited from the knowledge-base (presented in Chapter 3) in proposing an approach that could be used to simulate the process of FVM scanning and identify the most beneficial mobility algorithm for each given environment.

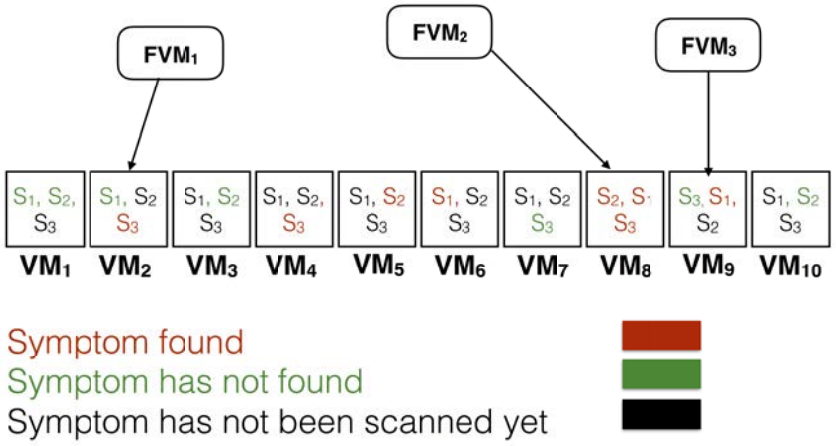


Figure 5.2: An example of a case study generated.

The proposed probabilistic approach is obtained based on probability theory and Bayes' theorem; Figure 5.1 illustrates the steps involved in the approach. It mainly benefits from information derived from the knowledge-base, such as the probability of seeing an exact set of symptoms when there is a malware infection or not. The information derived is used to develop a sample environment (or a case study, as shown in Figure 5.2) with a number of infected and non-infected VMs along with a number of symptoms, that can be seen on them in either case. A number of parameters need to be determined before developing the case studies and initiating the scan, such as the number of scanned VMs, the cost of false positives and false negatives, and the initial belief of a malware infection $P(Malware)$. The scan is then carried out on the given case study by a number of deployed FVMs, which are initialised with a specific mobility algorithm. During the scanning process, the initial belief of the infection $P(Malware)$ will be updated, according to the new evidence gathered, using Bayes' rule. The evidence

in this case is like confirming or denying the existence of a symptom S_i on a specific VM. Updating the initial belief after each step of the scan will help in determining which VMs are probably infected, given the set of identified symptoms (either found or not). Based on the VMs' status identified, the cost of the scan could then be determined (in terms of the scanning resources used, and the accurate identification of infections); therefore, the most beneficial mobility algorithm could be identified by a comparison of the resulting cost.

The reminder of this section presents the mathematical formulation of the approach; it can be described amply as follows:

Definition 5.1 (Set of all symptom sub-assignments)**.**

For a set c of boolean symptoms: $S_1, \dots S_n$ (which may be true or false), set of all symptom sub-assignments is defined as:

$$AS = \bigcup_{S \in P(c)} allAssignments(S) \tag{5.1}$$

Where:

- $allAssignments(\emptyset) = \{\emptyset\}$
- $allAssignments(\{S_i\} \cup S) = mapUnion(S_i = True, allAssignments(S \setminus \{S_i\})) \cup mapUnion(S_i = False, allAssignments(S \setminus \{S_i\}))$ where $S_i \in S$
- $mapUnion(b, \emptyset) = \emptyset$
- $mapUnion(b, \{A_i\} \cup A) = \{\{b\} \cup A_i\} \cup mapUnion(b, A \setminus \{A_i\})$ where $A_i \in A$

Given a configuration such as the one used to examine whether VMs are infected with the Sality malware family (as shown in Table 5.1), an example of a resulting set from Definition 5.1 could be: $\{S_7 = True, S_8 = False, S_9 = True\}$, which is a member of AS. After generating the set, we could then derive the following information, for all $\mathcal{S} \in AS$, by making use of the knowledge-base (presented in Chapter 3):

- $P(\mathcal{S} \mid Malware)$: is the probability of seeing $\mathcal{S}$ when there is malware.
- $P(\mathcal{S} \mid Benign)$: is the probability of seeing $\mathcal{S}$ when malware is not present

Therefore, during the scanning process, Bayes' theorem (mentioned in Chapter 2, p. 38), is used to calculate the degree of belief in a malware infection $P(Malware|\mathcal{S})$, taking into account the likelihood of evidence occurring in a VM, and the initial belief $P(Malware)$ as follows:

$$P(Malware|\mathcal{S}) = \frac{P(\mathcal{S} \mid Malware) \ . \ P(Malware)}{P(\mathcal{S})} \tag{5.2}$$

where $P(\mathcal{S})$, the probability of seeing a group of symptoms $\mathcal{S}$, can be calculated as follows (based on Equation 2.2, p. 39):

$$P(\mathcal{S}) = P(\mathcal{S} \mid Malware) \ . \ P(Malware) + P(\mathcal{S} \mid Benign) \ . \ (1 - P(Malware)) \tag{5.3}$$

As a result, depending on the value of $P(Malware|\mathcal{S})$, we can claim that there is malware on the VM scanned if $P(Malware|\mathcal{S}) \geq$ *a predefined threshold*. We will be using the value (0.5) as a threshold in the following experiment, based on some preliminary tests, where 0.5 gave us an appropriate separation between the two classes[1]. Therefore, we could describe the VM's status as follows:

$$P(Malware|\mathcal{S}) \begin{cases} \geq 0.5, & \text{there is a malware} \\ < 0.5, & \text{there is no malware} \end{cases} \tag{5.4}$$

Subsequently, we could say that the cost of false positive and false negative can be calculated based on $P(Malware|\mathcal{S})$; and therefore, the cost of inspecting a set of symptoms $\mathcal{S}$ on a VM using the proposed approach could be calculated as follows:

[1] A more thorough analysis could also be carried out to identify the best threshold for any given situation.

Definition 5.2 (Cost of inspecting a group of symptoms)**.**

Let the cost of false positive represented as $\mathcal{FPC}$, and the cost of false negative as $\mathcal{FNC}$ and the cost of reading $\mathcal{S}$ as $\mathcal{SC}$, the cost of identifying a group of symptoms $\mathcal{S}$ in a VM is:

$$\texttt{Cost}(\mathcal{S}) = \begin{cases} \mathcal{SC} + (1 - P(Malware|\mathcal{S})) \cdot \mathcal{FPC}, & P(Malware|\mathcal{S}) \geq 0.5 \\ \mathcal{SC} + P(Malware|\mathcal{S}) \cdot \mathcal{FNC}, & P(Malware|\mathcal{S}) < 0.5 \end{cases} \tag{5.5}$$

where:

- $P(Malware|\mathcal{S})$, refers to the degree of belief that there is an infection on the scanned VM, given the set of symptoms $\mathcal{S}$ which have been observed.
- $\mathcal{FPC}$, refers to the cost of incorrectly identifying a VM as infected.
- $\mathcal{FNC}$, refers to the cost of incorrectly identifying a VM as non-infected.

A conclusion can then be drawn based on the overall cost of the scan; and thus the most efficient mobility algorithm, which helps in finding the sets of symptoms with the lowest cost, could be identified. The following section details the architecture of the *the Mobility Algorithms Analyser* framework which was developed based on the probabilistic approach described earlier in this section.

5.4 The Mobility Algorithm Analyser Framework

In this section, we focused on evaluating the feasibility of the approach proposed in isolation from the cloud's environmental factors, such as data transfer, spin-up times or placement [28]. Therefore, we developed a proof of concept (referred to as The Mobility Algorithm Analyser Framework) to evaluate the proposed approach and to simulate the

FVM scanning process rather than using other simulation framework (e.g. [28]) which could introduce unnecessary complexities to our evaluation.

The Mobility Algorithm Analyser Framework was developed in Python based on the probabilistic approach presented in Section 5.3. The overall architecture of the Mobility Algorithm Analyser framework is shown in Figure 5.3, and it could be broken down into the following components:

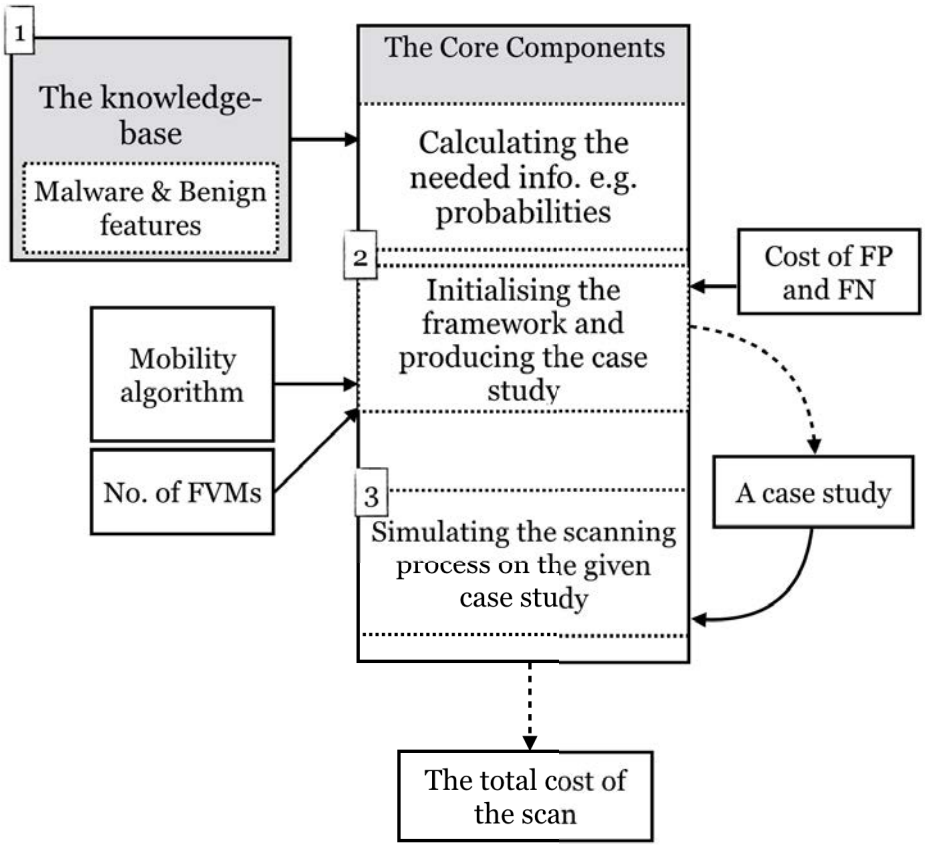


Figure 5.3: The Mobility Algorithms Analyser.

1. The knowledge-base. An overview of the process employed to develop the knowledge-base is described in (Chapter 3). The set of symptoms used is described in Section 5.2.

2. The Environment Modeller. It is comprised of two sub-components which perform the following tasks:

(i) Retrieving symptoms' probabilities from the knowledge-base in order to produce a realistic case study. This includes calculating the probabilities of seeing different sets of symptoms in case there is an infection or not.

(ii) Initialising the framework and producing the case study (as shown in Figure 5.2) based on the probabilities calculated in (i), and given a number of parameters that represent the scanned environment (e.g. number of VMs scanned, FVMs deployed, the cost of false positive and false negative, and the initial belief of a malware infection $P(Malware)$).

3. A discrete-time simulator. It is used to simulate the FVM's scanning process (according to an embedded mobility algorithm), using the case study generated by the *Environment Modeller* component. The simulator will update the initial belief of the infection $P(Malware)$, given the set of symptoms $\mathcal{S}$ that is identified during the scan on each VM. The process of updating the initial belief, and calculating the cost is described in detail in Section 5.3. We opted to implement this Python-based simulator not only because we want to facilitate the integration between the framework's components, but also because we were not aware of any available discrete-time simulation tool that could be used at the time of developing the framework presented.

At the end of the scan, the most beneficial mobility algorithm could be identified through a comparison of the resulting scanning cost when using each algorithm. The evaluation presented in Section 5.5 demonstrated this, where a number of mobility algorithms were compared, and the one that resulted in the minimum scanning cost was identified.

5.5 Evaluation

This section describes the experimental setup, results, and discussion of the evaluation conducted. The objective of this evaluation is to assess the feasibility of the approach through a comparison of multiple mobility algorithms, when scanning for different mal-

ware infections.

5.5.1 Experimental Setup

During the experiment, we have made the following assumptions:

1. We are not considering the case wherein a system state changes from infected to non-infected, or *vice versa* during a scan, because we assume we are dealing with only a snapshot of a system.

2. When an FVM scans for a specific symptom, it will eventually identify it.

3. We are assuming discrete time steps in which the scans of each symptom take the same length of time.

In addition to the above assumptions, it is important to state that the results presented depend on malware data examined during the experiment. We have used the most informative registry keys for each malware family, based on those mostly recorded by Symantec [169], and Microsoft [109] (as described previously in Section 5.2, and shown in Table 5.1). We have evaluated the proposed approach using a sample environment comprised of 10 virtual machines and 3 Forensic Virtual Machines, where each scans for a different symptom. We have looked here at changes performed on three registry paths, as from looking at the information provided by Symantec and Microsoft (e.g. in [108, 163]), it seems that the presence of a malware infection could be inferred in a system with as low as two symptoms (or registry paths); also, given the large number of malware families and variants a cloud provider needs to scan for, the minimum number of FVMs (which could sufficiently identify the early signs of malware infections) is expected to be allocated for each malware family. The evaluation is obtained through a comparison of the cost resulting from the scan using 4 different mobility algorithms, as the main aim of this chapter is to provide a way to compare mobility algorithms and nominate the most beneficial one. The algorithms[1] used in the experiment can be described as follows:

[1]The term *algorithm* is often used interchangeably in this chapter with the term *strategy*.

1. **Strategy 1:** In this strategy, a VM will be chosen randomly from a subset of VMs; the subset contains a predefined percentage of VMs, which have the highest ratio of the discovered symptoms, to the total number of symptoms we are scanning for. In the documented experiments we have obtained the subset by taking the top 40% VMs that have the highest ratio. This strategy is inspired by the mobility algorithm proposed by [69], which has been described later on in this section (in **Strategy 4**), with setting the scaling factor (ω) to 0.

2. **Strategy 2 and 3:** In both, each FVM is assigned different pre-programmed priorities, and it chooses its next scanning targets based on those priorities. The priorities in both strategies are set based on the probability of having malware, given a group of identified symptoms $\mathcal{S}$; where this has been calculated based on information gathered from the knowledge-base developed in Chapter 3. Therefore, when an FVM that scans for symptom S_p is informed that S_m and S_n have been found on two different VMs (v_i and v_j respectively), the FVM then has to check its pre-set priority to determine which VM has the symptom that (when combined with the symptom that it is scanning for) results in the highest probability between the two, $P(Malware \mid S_p \cap\ S_m)$ and $P(Malware \mid S_p \cap\ S_n)$[1]. It will then set this VM as the next scanning target. The only difference between those two strategies, is that the former will set its next scanning target based on the probability of having malware, given only two symptoms, whereas the latter will consider all three symptoms instead. However, if the symptoms considered have not been inspected yet, the VM target will be then be chosen completely at random.

3. **Strategy 4:** Harrison et al. [69] proposed a mobility algorithm which is based on calculating the following for every $v \in \mathcal{A}$, where $\mathcal{A}$ is a subset of VMs (that represent

[1]We have also considered the set of symptoms that minimises $P(Benign \mid \mathcal{S}\ )$ at the same time.

the potential scanning targets):

$$F(v) = \sum_{i=1}^{K} \frac{Disc(c,v)}{size(c)} val(c) + \omega\ [CurrentTime - LastVisited(v)], \tag{5.6}$$

where:

- $\frac{Disc(c,v)}{size(c)}$ is the ratio of the number of discovered symptoms that belong to the configuration c, to the total number of symptoms in the same configuration; an example of a configuration is presented in Table 5.1.
- $val(c)$ is the severity value of configuration c assigned by a security expert.
- $[CurrentTime - LastVisited(v)]$ is the time elapsed since a VM has been scanned or visited by any FVM.
- ω is the *Loneliness Parameter* where $\omega > 0$, and the lower the number we set this parameter to, the less important the timing of the last visiting will be.

After all $F(v)$ are calculated based on Equation 5.6, a pre-determined percentage of VMs with the top $F(v)$ values will be retrieved, and given this reduced set, a random VM will be nominated for the scan.

We tested the mobility algorithms mentioned on four malware families, and three malware types chosen randomly from the malware database. The results of the experiment were obtained by conducting a number of trials. On each trial the framework was used to perform the following: (i) generate a large number of case studies (based on the probability of an infection (i.e. $P(Malware)$), and the distribution of symptoms whether there is an infection or not); and (ii) simulate the FVMs' scans on each case study generated for a fixed number of steps; and (iii) average the cost over all the scans performed to attain a more robust results.

The `Cost` function (presented on Equation 5.5) is calculated based on the previously mentioned assumptions and given the following values, 15 and 5 for $\mathcal{FNC}$ and $\mathcal{FPC}$

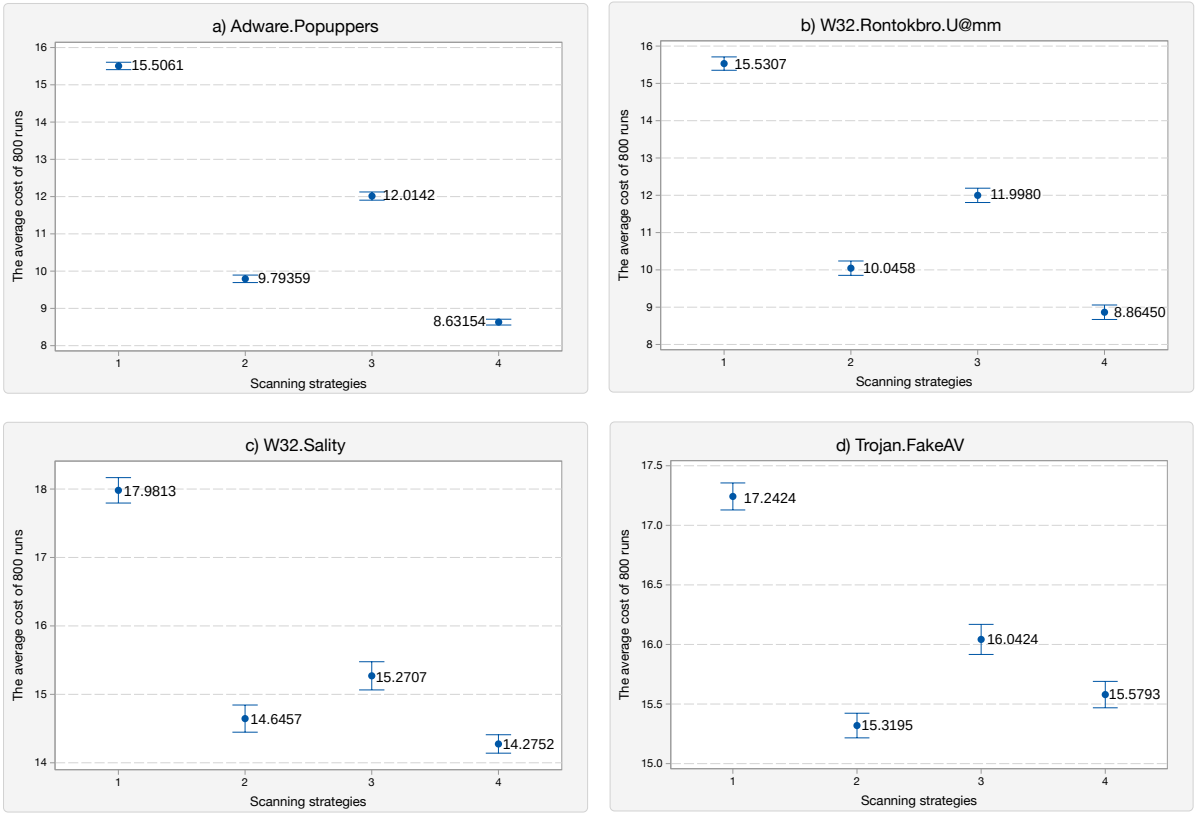


Figure 5.4: The 95% CI of the means for each malware family

respectively; assuming a case, where failing to report a malware correctly is more costly than reporting an infection where there is none. It is worth noting that the exact values were chosen here entirely arbitrarily, and an organisation could consider more realistic values, based on quantifying different type of impacts; e.g. based on the financial impact of false positives and negatives.

5.5.2 Results

In each test, FVMs were initialised with one of the mobility algorithms presented on p. 95. They were then used to scan for a possible infection with a malware family (or type), by identifying a set of distinctive registry key changes (the set of registries used are presented in Table 5.1, p. 86). Based on a comparison of the 95% *confidence interval* (CI) of the cost means, the most beneficial mobility algorithm could then be identified in each malware family and type; the cost recorded is shown in Figure 5.4 and Figure 5.5. In order to make

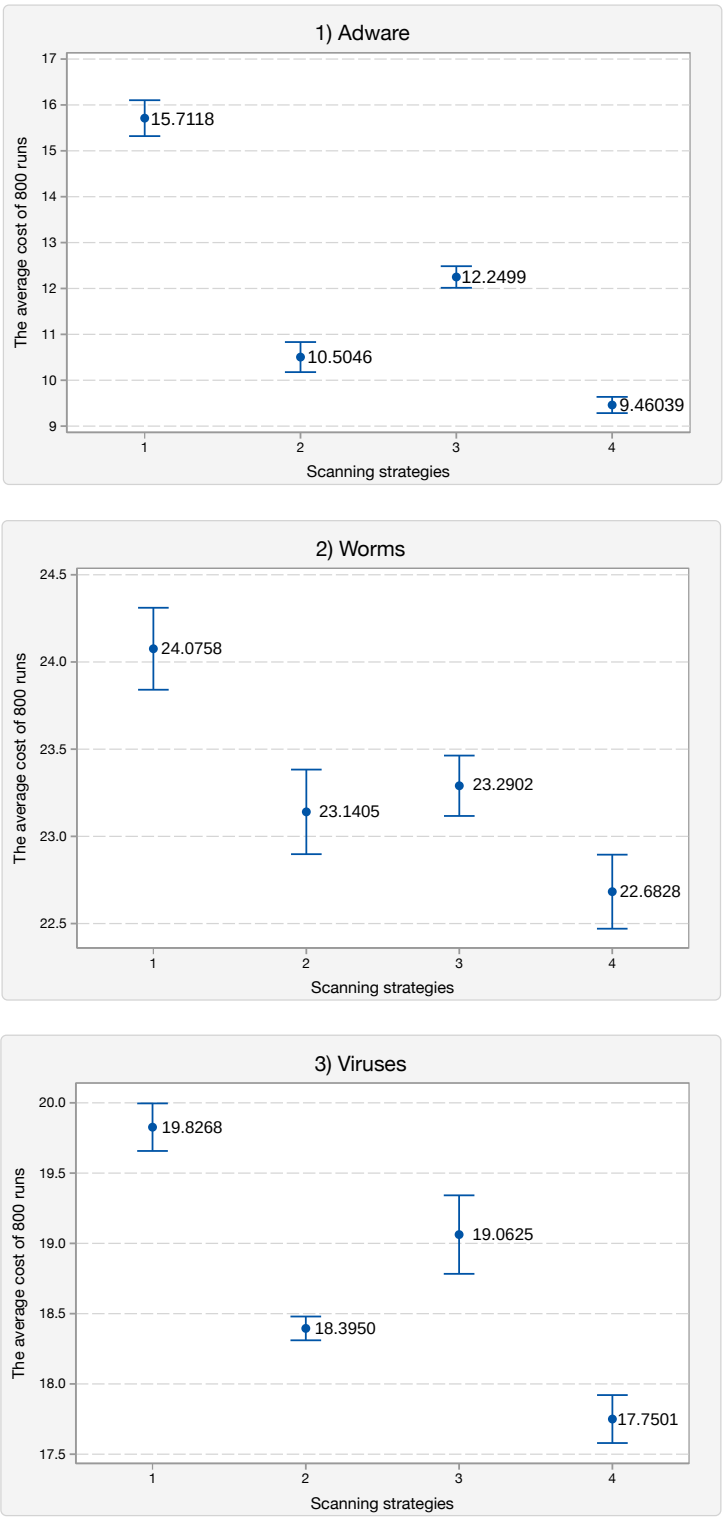


Figure 5.5: The 95% CI of the means for each malware type

Table 5.2: Malware families used in the experiment

Malware Family	# Samples	Malware Type
Adware.Popuppers	167	Adware
W32.Rontokbro@mm	210	Worm
W32.Sality	109	Virus
Trojan.FakeAV	102	Trojan

sure that the graphs shown and the confidence intervals calculated represent a reasonable estimate, we calculated the CIs based on a number of trials. In each trial, we averaged the cost over a large number of runs (as described on p. 96), instead of calculating the CIs based on a number of single runs. More thorough evaluation of the probabilistic model proposed could also be conducted using probabilistic model checking techniques; however, this has been left for future work as discussed later on in Section 5.7.

Using the Mobility Algorithms Analyser framework, we firstly tested the FVMs with 4 malware families; Table 5.2 shows the tested malware families, along with the number of malware samples used during the experiment, in addition to the type of malware that each family belongs to. During the testing of malware families, we recorded the results of 20 trials. In each trial we simulated 800 scans. From Figure 5.4, it can be seen that there was a considerable difference in the resulting scanning cost when different mobility algorithms were used. Also, it can be seen that none of the recorded confidence intervals of the algorithms showed any overlap when each malware family was tested. This indicates that there was a significant difference in the performance between the strategies. Based on Figure 5.4, we can conclude that the optimum algorithm for `Adware.Popuppers`, `Rontokbro` and `Sality` is *Strategy 4*, as it gave the lowest scanning cost, whereas *Strategy 2* showed the lowest scanning cost when used to scan for `Trojan.FakeAV`. Figure 5.4 also shows that *Strategy 1* always resulted in the highest cost.

We have also evaluated the strategies given different malware types, in order to determine whether the most efficient strategy could vary. Table 5.3 shows the malware types tested, the families tested in each type, along with the number of malware samples used during the experiment. The distinctive symptoms used during the FVMs' initialisation

Table 5.3: Malware types used in the experiment

Malware type	**families tested within this type**	**# Samples**
Adware	Adware.Popuppers	167
	Adware.Clkpotato!gen3	32
	Adware.Istbar	26
	Adware.Slagent	26
	Adware.ZenoSearch	14
Worms	W32.Rontokbro@mm	210
	W32.Spybot.Worm	196
	W32.Benjamin.Worm	88
	W32.Mabezat.B!inf	35
	W32.SillyFDC	25
Viruses	W32.Sality	109
	W32.Virut	97
	W32.Whybo!inf	88
	W32.Licum	27
	W32.Xpaj.C	21

when testing malware families, were also used when testing the corresponding malware type. This relies on the assumption that all malware families of the same type share similar symptoms. The results shown on Figure 5.5 were obtained by recording the cost of 12 trials, where in each the cost was averaged over 800 scans.

Figure 5.5 shows that the most efficient mobility algorithm for all the tested malware types tended to be *Strategy 4*, while the highest cost was observed when the scan was performed using *Strategy 1*. It can also be seen in the case of `Adware`, that the dependency between the symptoms is relatively strong, which might induce the gap between the results for *Strategy 1* and both strategies *2* and *3*. The gap observed between *Strategy 1* & *4* might be due to the disparities in the scanning coverage. As the coverage obtained by *Strategy 1* is expected to be limited, due to that this strategy guides FVMs to scan those VMs which have more discovered symptoms, unlike *Strategy 4* that aims to avoid *lonely VMs*, which have not been inspected for a long time.

5.6 Comparison with the Related Work

In the following section we review the literature that is directly related to our work, and we indicate how this work differs.

Since Forensic Virtual Machines were introduced by Harrison et al. [69], a number of mobility algorithms, that define the movements of FVMs from one VM to another, have been introduced. For example, FVMs can be prompted to scan VMs randomly, according to a specific order, or based on a dynamic mobility algorithm.

Harrison et al. [69] introduced the mobility algorithms presented previously in Equation 5.6 (page 96). The proposed mobility algorithm aims mostly to reduce cases where a VM is left for a long time without being scanned. Another mobility algorithm has been proposed by Alshamrani et al. [9], where the target VM is chosen based on the time since the VM was last scanned, in addition to the weight assigned to the VM.

Our work in this chapter mainly addresses some unanswered questions regarding the development of a technique that helps to choose the most efficient mobility algorithm to be used by FVMs during the scanning process, thereby nominating in advance the one that can minimise the cost of the scan for the environment in question. This helps in promoting a prudent use of the available scanning resources, as the creation of FVMs is a resource-intensive process [23]. We have also introduced two mobility algorithms, represented in *Strategy 2* and *3* (as described in p. 95) to be used in the evaluation phase of the framework proposed.

5.7 Conclusion

Before initialising an FVM, it is important to be able to recognise the most beneficial mobility algorithm, which helps in identifying the most critical set of VMs with the minimum consumed cost. This chapter presented a probabilistic approach for nominating the most cost-wise mobility algorithm for each given environment. The chapter has demonstrated the feasibility of the proposed technique, where the most cost-wise mobility algorithm for

each given environment could be identified. The evaluation of the approach was based on a comparison of the 95% confidence intervals for the cost means of a number of mobility algorithms; however, in future research we aim at conducting a more thorough evaluation of the model proposed by making use of probabilistic model checking techniques (e.g. PRISM [95]). There are also other cases we have not considered which will demand future work. For example, we have not considered the case when a system state changes from infected to non-infected or *vice versa* during a scan. We have also, for the purpose of simplification, considered only those cases in which all the VMs on the system's current snapshot are infected with the same family or type of malware. However, considering the size of the cloud, a VM could be subject to multiple attacks; having multiple malware families or type on the same time should therefore be taken into account.

CHAPTER 6

A MARKET-BASED APPROACH FOR DETECTING MALWARE IN THE CLOUD VIA INTROSPECTION

6.1 Introduction and Motivation

In the previous Chapter, we targeted some unanswered questions regarding optimising the scanning performance of a recently proposed malware monitoring technique, which targets the cloud environment (i.e. Forensic Virtual Machines). This technique leverages Virtual Machine Introspection technology, which enables the monitoring of VMs externally, in order to carry out a fast and light-weight scanning of the targeted VMs.

As described earlier in Chapters 2 and 5, VMI-based malware monitoring techniques offer significant advantages over traditional malware scanners: (i) they increase the level of security as they isolate the inspected system from the malware scanner, and (ii) they limit the usage of resources and the performance impact on the inspected VM, as external monitoring of a VM's internal state is performed. Moreover, the fact that they can detect behavioural symptoms (which are shared among a number of malware families), as opposed to matching signatures on traditional AV solutions, makes them an ideal solution for monitoring early signs of infection. This feature also provides them with the capability of detecting unseen malware threats, as well as making the scanners much less demanding for frequent updates.

However, the reduced usage of resources could be at the expense of wrongly flagging more VMs as infected. This is because such techniques are based on identifying symptoms that exist in both malicious and normal behaviour, but in different proportions or combinations; where in some cases a clear-cut distinction between the two behaviours is not easy to identify. Thus, although these techniques can provide a means for detecting early signs of infections, they do not provide definitive identification results. In contrast, signature-based malware detection systems are known to accurately identify known malware threats with an extremely low rate of false alarms due to their usage of unique signatures [67], but at the cost of using a relatively high percentage of the VM's resources during the scan.

This chapter provides a way to integrate VMI-based light-weight scanners (e.g. FVMs) to improve the efficiency of malware detection on the cloud without sacrificing the accuracy of detection. This outcome could be achieved by addressing the issue of balancing the trade-off between scanning performance (in terms of the scanning resources used) and accuracy (in terms of the false alarms generated), when detecting malware infections on a multi-tenant cloud environment (where resources, e.g. VMs, are delivered to multiple cloud users or customers [49]).

Towards fulfilling this objective, this chapter proposes an early prioritisation approach, which consists of two layers of protection (i.e. light-weight and full malware scanning) for a more in-depth scan of customers' VMs. The proposed approach benefits from the fast and light scanning performance of FVMs to identify, at any given time, a pre-defined percentage of VMs that are most likely to contain malware. It will then trigger a full malware scan on this set of VMs. The novelty of this technique is that it utilises a market mechanism to guide the full scanning process, based on the criticality of the pieces of information gathered by the distributed monitoring carried out using the VMI-based light-weight scanners. The use of signature-based detection methods here might imply that the approach presented inherits the high rate of false negatives due to the fact that such malware detection methods are known for their inability to recognise malware

infections that have not previously been seen; however, this issue has been mitigated by combining two different detection techniques that are expected to complement each other (as discussed later on, see p. 118): signature-based (i.e. full malware scanning), and behaviour-based (i.e. FVMs).

We proceed as follows: Section 6.2 describes the proposed approach in detail. The results of the experiment along with the evaluation are then presented in section 6.3. Afterwards, Section 6.4 reviews the related work and describes how the contribution presented differs. This chapter then discusses the research outcomes and outlines the conclusion in Section 6.5.

The contents of this chapter are based primarily on one of our contributions to the literature (Alruhaily et al. [8]).

6.2 The Market-Based Prioritisation Approach

The proposed approach relies on a market-based mechanism to prioritise the VM scanning process. Such a mechanism promotes both an optimal resource allocation [102], and a highly distributed operation [191], which in our context leads to a scalable prioritisation of limited FVMs to scan a larger number of VMs. It aims to balance the trade-off between the scanning performance and the accuracy of detection by achieving the following subgoals:

(i) maximising the number of VMs scanned using light-weight scanners (i.e. FVMs);

(ii) minimising false alarms generated, by identifying the most critical VMs and scanning them thoroughly.

Figure 6.1 depicts the high-level architecture of the proposed market-based prioritisation approach; the approach consists of two protection layers, whereby a bidding process is performed on each.

Consider a set of Virtual Machines $\mathcal{V} = \{v_1, v_2, ..., v_m\}$ and a set of FVMs, $\mathcal{F} = \{f_1, f_2, \cdots, f_n\}$, where $n < m$. At each time step, information gathered previously by

the set of FVMs at the *first protection layer* will be used to guide the full malware scan initiated at the *second protection layer*, whereby each VM submits a sealed-bid, to be scanned thoroughly by an AV. To minimise the number of costly scans, only a pre-defined percentage (μ) of the VMs most likely to be infected will be scanned using AV instances. In the meantime, the remaining set of VMs, which have not been assigned to be scanned by AV instances, will submit a sealed-bid, requesting to be inspected further by FVMs, in order to guide the full malware scan at the next time step.

In our scenario, the relationship among the VMs is competitive and non-cooperative; therefore, sealed bidding (described in Chapter 2, p. 41) has been used in order to promote efficient allocation of the scanning resources. The efficient allocation of the resources is due to the fact that decisions are made instantaneously, without resulting in a considerable overhead during the allocation process [66]. An important feature of bidding is their dynamic pricing-discovery property where the price is determined based on the criticality of the buyers' status (i.e. VMs). For example, when an FVM identifies a symptom on a VM, where this symptom by its own is not definitive, but it can be associated with a malware infection, this will increase the bidding price of that VM. Consequently, the VM will be more likely to be scanned in the second protection layer by an AV instance, in order to confirm whether or not the VM is infected. On the other hand, the absence of such a symptom on a VM can decrease its bidding price; meaning it will be less likely to be scanned by an AV instance at the second layer of protection.

The remainder of this section describes in detail the bidding process at the two protection layers.

First Protection Layer: scanning VMs with FVMs.

The monitoring task is subdivided between FVMs, where each FVM scans for a single symptom. It is thus of the utmost importance to identify which symptom should be inspected next on each VM. This is crucial as it will lead to an optimal allocation of the light-weight scanning resources, leading to an efficient use of the AV instances which are

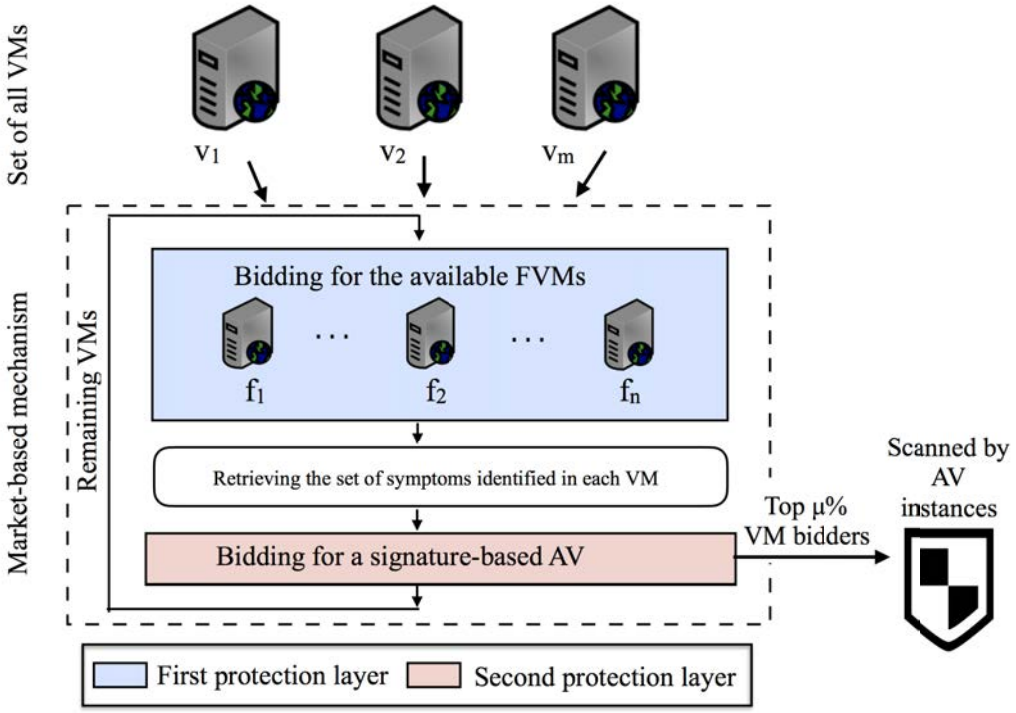


Figure 6.1: High-level architecture of the market-based prioritisation approach.

influenced by the information gathered at this layer. Therefore, before an FVM moves to a new VM target, it will enter into a deciding state, where every VM submits a sealed bid for each type of FVMs. Given the set of FVMs, $\mathcal{F}$, where each scans for different symptoms (i.e. f_1 scans for S_1, f_2 scans for $S_2 \cdots$, and f_n scans for S_n), the bid submitted by each $v \in \mathcal{V}$ to each f_j is given as follows:

$$Bid(f_j, v) = E(v) + \lambda \cdot I(X; S_j | Z = \mathcal{S}_v) \cdot t(v), \tag{6.1}$$

where:

- $E(v)$ refers to the expected impact when the VM becomes infected. It is estimated based on $P(Malware|\mathcal{S}_v)$, the probability of having a malware infection given the set of identified symptoms $\mathcal{S}_v$, and $Im(v)$, the impact resulting when the VM is compromised and becomes infected. This value $Im(v)$ can be used to quantify different types of impacts (e.g. financial and reputational impacts); in the current

scenario, it was assumed to take a value between '0', representing no impact, and '5', representing the maximum impact. The expected impact $E(v)$ is then given as follows:

$$E(v) = P(Malware|\mathcal{S}_v) \cdot Im(v), \tag{6.2}$$

- $x \in$X (i.e. X= $\{Malware, Benign\}$).
- S_j is the new symptom inspected by the corresponding f_j (i.e. $S_j \in \{True, False\}$).
- $\mathcal{S}_v$ is a set of previously scanned symptoms on each $v \in$ V (either found or not found), one example might be: $\{S_2 = True, S_5 = False, S_8 = True\}$. The sets of the scanned symptoms are retrieved before initiating the bidding at each time step.
- $I(X; S_j|Z = \mathcal{S}_v)$ (Definition 2.4, mentioned in p. 40) is the mutual information of the two random values X and S_j, except in this case we are always conditioning on observing the event $Z = \mathcal{S}_v$. We refer to this mathematical expression as '*event-specific conditional mutual information*'; it denotes the amount of information obtained between the two variables X and S_j given the observation of the third random variable $Z = \mathcal{S}_v$. It can be written mathematically as follows:

$$I(X; S_j|Z = \mathcal{S}_v) = \sum_{y \in S_j} \sum_{x \in X} P(x, y|\mathcal{S}_v) \cdot log \ \frac{P(x, y|\mathcal{S}_v)}{P(x|\mathcal{S}_v)P(y|\mathcal{S}_v)} \tag{6.3}$$

- $t(v)$ is the time elapsed since the VM was visited by any FVM.
- λ is a scaling factor where $\lambda > 0$; it is used to adjust the importance of the information gathered (in terms of its relatedness and recentness) with the impact associated when a VM being compromised .

After the bids are collected at each time step, the price of an FVM type is determined, based on the highest bid submitted; the VMs with the highest bid will then be scanned

by that type of FVMs. The symptoms scanned at this layer will help prioritise the full VM scan initiated at the next step.

Second Protection Layer: scanning the VMs that are most likely to be infected with AV instances.

On this layer, VMs will bid for a full malware scan, which will confirm whether they are infected. To minimise the costly use of resources, only a predefined VM percentage (i.e. μ) of the most critical VMs will be thoroughly scanned by an AV instance. The percentage μ could be adjusted by the Cloud Service Provider to fulfil two primary goals:

(i) maximising the scanning coverage by the available FVMs; and

(ii) minimising cases where a costly full malware scan is triggered on a non-critical VM, thus minimising the unnecessary usage of the scanning resources.

At this layer, every $v \in$ V will submit a sealed-bid requesting a thorough scan by a signature-based AV instance, as follows:

$$Bid(AV, v) = P(Malware|\mathcal{S}_v), \tag{6.4}$$

where:

- $P(Malware|\mathcal{S}_v)$ is the probability of having a malware infection given the set of identified symptoms, $\mathcal{S}_v$; the probability is calculated according to Bayes' Theorem (Theorem 2.3, mentioned in p. 38).

The remaining set of VMs that have not been scanned using signature-based AV instances will then be inspected further by re-entering the bidding process carried out at the *first protection layer*.

6.3 Evaluation

This section describes the experimental setup, results, and discussion of the experiments conducted. The objective of this evaluation is to assess the performance of the proposed market-inspired prioritisation mechanism by checking whether it balances the trade-off between the consumption of resources and the false alarms.

6.3.1 Experimental Setup

As in Chapter 5, the focus here is on evaluating the approach proposed in isolation from the cloud's environmental factors (e.g. data transfer, spin-up times, or placement [28]). Therfore, a proof of concept was developed, which extended the framework from our previous work (presented in Chapter 5) with the proposed market-based approach. In addition to the assumptions made, when developing the simulation, on p. 94 (i.e. the discrete time steps, unchanging status of the infection, and FVMs' ability to identify the existence of symptoms), we have also assumed here that the signature-based AV is up-to-date, and that a signature of the malware exists in the AV database of signatures. This additional assumption is based on our focus on reducing the false alarm error in the evaluation presented; however, in terms of the false negative error which signature-based AV solutions are usually susceptible to (i.e. failure to identify an infection when in fact there is), it is expected that the two approaches complement each other as described later on in this chapter (p. 117).

The proposed market-based approach was compared to the use of VMI-based light-weight scanners alone (i.e. FVM scanners) under the same settings. In particular, the simulator was initialised with 100 VMs, and 8 FVM types, with each type composed of 6 FVM instances. The same set of symptoms was also used, represented on the 8 most informative registry paths, accessed to modify, add, delete or read a subkey value when infected with a variant of the *W32.Sality* malware family, which has long been ranked as one of the top 20 malware families in [166]. The most informative registries

Table 6.1: Most informative registry paths accessed when infected with *W32.Sality*.

Symptom ID	**Registry paths accessed**
S_1	HKLM\SOFTWARE\Microsoft\Security Center
S_2	HKLM\SOFTWARE\Microsoft\Security Center\Svc
S_3	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
S_4	HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List
S_5	HKLM\SYSTEM\CurrentControlSet\Services\Winsock\Parameters
S_6	HKLM\SYSTEM\CurrentControlSet\Services\Winsock\Parameters\Namespace_Catalog
S_7	HKLM\SYSTEM\CurrentControlSet\Services\Winsock\Parameters\Protocol_Catalog
S_8	HKLM\SYSTEM\CurrentControlSet\Services\Winsock\Parameters\Protocol_Catalog\Catalog_Entries

have been determined by utilising the latest Scikit-learn Python library [149], where a feature selection process was performed using the *ANOVA F-value* feature selection method; Table 6.1 lists the set of the symptoms used. The experiments were also based on a 0.11 probability of malware infection. This was determined based on the most recent report obtained by [13] when conducting the experiments. The value of μ was set to 1, meaning that only 1% of the VMs most likely to be infected would be scanned by AV instances at each time step[1].

We have used the following measures to assess the performance of the detection: the *True Positive Rate* and *False Positive Rate* (which are referred to as the *Detection Rate*, and *False Alarm Rate*, respectively, in the intrusion detection community [150]). We were mainly interested in confirming the infection status of the VMs most likely to be infected, in order to reduce the false alarm; however, the results of the two aforementioned metrics have been recorded due to the fact that there is a known trade-off between them [61]; the metrics are given as follows:

[1]Ceil function was used to ensure that at least 1 VM is scanned at each time step.

$$
\begin{aligned}
False\ Alarm\ Rate &= \frac{FP}{number\ of\ negative\ samples} = \frac{FP}{TN+FP},\ and \\
Detection\ Rate &= \frac{TP}{number\ of\ positive\ samples} = \frac{TP}{TP+FN},
\end{aligned}
\tag{6.5}
$$

where:

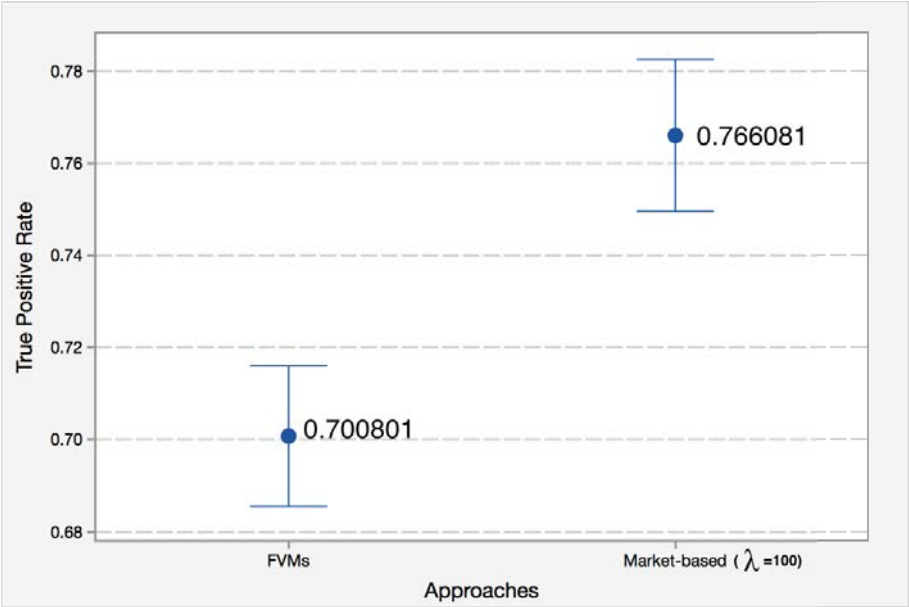
- *True positive (TP)*: Number of infected VMs correctly labeled as infected.
- *True negative (TN)*: Number of not infected VMs correctly labeled as not infected.
- *False positive (FP)*: Number of VMs not infected incorrectly labeled as infected.
- *False negative (FN)*: Number of infected VMs incorrectly labeled as not infected.

6.3.2 Results

The main objective of the evaluation was to assess the performance of the proposed approach, and to determine whether it balances the trade-off between the consumption of resources and the accuracy of detection (in terms of false alarms). As a step towards this objective, this section presents the results of testing the proposed approach with respect to two dimensions: (i) the accuracy of the detection (represented on the disparity of false alarms generated), and (ii) the consumption of the VMs' resources.

Accuracy of the Detection

The detection performance of the proposed approach was evaluated through a comparison of the 95% confidence interval (95% CI) for the mean of the *Detection Rate* and *False Alarms Rate*, derived by simulating the scanning process of the proposed market-based approach, and the light-weight monitoring approach alone. Figure 6.2 was obtained by recording the results of 15 trials, whereby in each trial, the metrics averaged over 25 completed scans.



(a) Detection Rate

(b) False Alarm Rate

Figure 6.2: The 95% CI of the means.

The fact that there is no overlap in the 95% confidence interval for the mean of both documented rates, demonstrates that with statistical significance, a lower rate of false alarm and a higher detection rate could be achieved using the proposed approach, as shown in Figure 6.2(a) and Figure 6.2(b), respectively, with as low as only 1% usage of the heavy-weight scanning resources at each time step. This indicates that the proposed market-inspired prioritisation approach managed to guide the full scanning process (represented on the 27 AV instances) to confirm the infection status of those VMs that exhibited suspicious behaviour. Consequently, balancing the trade-off appropriately, by accurately identifying malware infections on these suspicious VMs, while promoting lower consumption of the cloud's VMs resources as a limited number of the heavy-weight scanning have been performed.

The heavy-weight scanning can also be guided to scan those VMs which are considered as more important (e.g. from an economic point of view), by reducing the value of λ in Equation 6.1. However, as our aim in this chapter was to make the best use of the small scanners to confirm the infection status of those VMs which have high probability of being infected (thus reducing the false positive generated), this part of the evaluation has been left for future work as discussed later in Chapter 7, p. 125.

Consumption of VMs' Resources

In order to check whether a reasonable balance had been achieved using the proposed approach, we compared the detection rate derived from the proposed market-based approach, with the detection rate expected to be achieved by using AV solutions alone based on the consumption of the same amount of VMs resources (in terms of CPU usage). We have assumed a CPU consumption of 19% and 0.3% per VM, when using an AV instance and an FVM, respectively. The choice of 19% CPU consumption is based on the fact that it was the minimum CPU consumption achieved according to [44], where they present a comparison between three leading AV solutions running inside a VM. While in the case of FVMs, the significantly lower consumption of resources is due to the fact that they

adopt a memory mapping approach to access the current status of a VM, this is expected to result in a considerably low performance impact on the inspected VMs [158], and a minimum amount of the VMs resources are consumed in that case.

In the previous section, the proposed approach was initialised with a value of μ=1%. Therefore, on average 27 full AV scans, and 648 FVMs scans were performed to detect malware infections on all the available VMs. This means that by using the amount of resources which was expected to be consumed by approximately 38 AV scanners, the proposed approach was able to achieve a detection rate of 0.766081 (as shown on the previous section). By using the same number of AV scanners to simulate performing a full malware scan of a random set of VMs, we were able to achieve a detection rate of only 0.333333 (averaged over 1000 runs). This indicates that by scanning the target VMs using AV solutions alone, a larger number of full scans was needed, and thus more resources are expected to be consumed in order to achieve a similar detection rate.

The low usage of resources that characterises the proposed approach helps in performing a large scale scanning, while keeping the consumption of the resources relatively low compared to that of using heavy-weight detectors only.

6.4 Comparison with the Related Work

In the following section we review the literature on malware detection in the cloud that is directly related to our work, and we indicate how this work differs.

A number of security vendors propose using light-weight in-VM agents to collect information and to facilitate the VM scanning process, while the heavy operations will be shifted to a scanning engine deployed on a dedicated VM (as discussed in Chapter 2, p. 33). Such an approach is susceptible to manipulation by an attacker, due to the lack of isolation, as protecting a VM is heavily dependent on the light-weight agent installed inside the VMs [75]. Therefore, researchers propose out-of-the-guest, VMI-based light-weight scanners to inspect the cloud's VMs from the outside [63, 92, 86].

VMI-based light-weight scanners provide a means to monitor early signs of infections; however, no internal change in the inspected VM is performed by such tools (as it is based on monitoring VMs passively as described in Chapter 2, p. 33). This can ensure the integrity of the inspected VM, even when those tools have been compromised; however, it means that for such a tool to be beneficial it needs to be integrated in a way that boosts the efficiency and effectiveness of the detection system.

Fischer et al. [57] proposed using a system comprising light-weight and heavy-weight detection engines which detect abnormal activities on the hosted VMs using introspection technology and machine learning-based methods. Although the proposed system provides the cloud infrastructure with the capacity to detect stealthy threats, and offers the required level of isolation, it relies mainly on behaviour-based monitoring approaches to detect threats, which makes it vulnerable to the high false alarms rate; it can also miss-detect a number of threats that are activated according to a specific argument or time (e.g. time-bomb malware). This problem has been addressed by utilising signature-based malware detection systems to detect such threats and to identify accurately known malware behaviour, therefore limiting the number of false alarms. Furthermore, our work proceeded one step further by providing a way to allocate the heavy-weight scanners efficiently using a market-inspired mechanism.

There are also other VMI-based approaches that are concerned with identifying infections accurately by using in-memory signatures (e.g. [174]), as opposed to identifying behavioural symptoms; however, these are beyond the scope of this work as using such techniques alone means that we could still inherent the limitations exist in signature-based detection systems (e.g. they cannot detect unknown malware infections).

6.5 Conclusion

This chapter has proposed a novel, market-inspired prioritisation approach, which utilises light-weight cloud-based scanners to guide the full VM malware scanning process, thus

promoting lower consumption of the cloud's VM resources, while accurately identifying malware infections. As the two layers of protection implement different detection techniques, they are expected to complement each other. The light-weight scanners, which perform external monitoring, will identify those VMs that need to be thoroughly scanned, without significantly affecting their performance. They will also ensure the integrity of the AV instances installed. Conversely, signature-based AV instances are used to confirm the infection status, due to their ability to accurately identify known malware infections. In order to reduce the overhead and complexities that might result from the bidding process in the market-based mechanism, bidding could be relaxed and simple prioritisation techniques could be used in cases where the number of VMs scanning candidates is dramatically reduced and choices become straightforward. The evaluation demonstrates the feasibility of the approach in terms of balancing the trade-off between scanning performance and the accuracy of the detection.

Although cases where VMs are infected with a malware whose signature is not yet added to the database (e.g. polymorphic and zero day malware) were not considered in the experiments, the proposed approach benefits from two layers of protection that implement different detection techniques: signature-based and behaviour-based (i.e. FVMs); these protection layers are expected to complement each other. Thus, if we encounter the case where a full malware scan is trigged on a VM, based on the malicious features identified, while the scanning result of the AV shows that the targeted VM is not infected, further analysis of the VM is still needed in order to identify the reasons behind the malicious activities observed; especially if the probability $P(Malware|\mathcal{S})$ is found to be very high.

Forensic Virtual Machines has been used here as a case study, however, other light-weight VMI-based monitoring approaches could be used as an alternative; the cloud provider could then adjust the heavy-weight scanning usage (i.e. μ) appropriately, so a lower amount of resources would be consumed without sacrificing the accuracy of the detection. In addition, when performing the experiments all FVMs were treated as idle, where in each bidding round all of them were expected to participate on the current

bidding process. This is based on the assumption that FVMs are assigned a similar Permissible Time to Stay (PT2S) (described in Section 2.4.2), and there is a negligible difference on the time spent to identify the symptoms (as mentioned in Subsection 6.3.1). However, when this is not the case, a time-based bidding strategy could be adopted, where each bidding round is performed within a given time frame, and only idle FVMs will participate in the current bidding process.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The generation and spread of malware is a major concern in computer systems and networks. The danger associated with such a threat may result not only in a financial loss, but also threaten the life of individuals; for example, a data breach could expose sensitive data of individuals to identity theft and fraud. In addition, an infection which locks or wipes data could incur a great cost and irretrievable damage (e.g. wiping the individual's medical records as in *WannaCry Ransomware* [112]). Therefore, researchers have endeavoured to detect and mitigate such a threat by introducing methods and tools that facilitate the process of analysing, detecting and preventing malware infections.

This thesis has addressed some open and unanswered questions in the field of malware detection. Therefore, a number of contributions have been proposed and evaluated in this thesis as solutions to the research questions (previously stated in Chapter 1, p. 8). We proceed as follows: Section 7.1 elaborates on how the solutions and the proposed approaches in the previous chapters have answered the research questions. The possible future research directions are then highlighted in Section 7.2. The thesis is then concluded in Section 7.3 with a brief summary.

7.1 Discussion and Reflection

The following section reflects on the contributions made, and revisits each research question stated in the first chapter of this thesis, and thereby determines whether the overall objectives have been met.

Question 1: What are the reasons behind the misclassification of malware? In Chapter 4 we carried out a number of experiments to answer this question; in particular, we classified malware grouped by year of discovery, in addition to grouping them into malware variants. Based on our experiments, we found no significant, long-term correlation between the time since a classifier was built and the misclassification rate of new malware (i.e. after a small initial drop off, the behavioural changes in the new malware did not reduce the accuracy of the classifier). Our research also showed that most of the behavioural changes that affected the classification rate of malware could be traced back to particular malware variants, and in most cases these changes are not replicated across malware families. This finding means that we would see a single variant that was hard to detect in a family of malware in which the other variants could be reliably detected.

We have also found that the misclassifications were mostly due to the adoption of anti-virtualisation techniques, or to the fact that particular variants were looking for a specific argument to run, or to the fact that some variants were actually considered as corrupted files. While removing all examples of misclassified corrupted malware from our dataset would have been possible, we note that no other work on malware classification does this, so removing these samples would not reflect other work.

The results presented could help the reader to interpret other papers that present the detection rate of a machine learning classification system as their main result. Furthermore, our work sheds light on how these detection systems will perform over time, and why some malware avoids detection.

Question 2: How can malware scanning performance of Forensic Virtual Machines be optimised ?

In Chapter 5 we proposed a probabilistic approach which identifies the most beneficial mobility algorithm to be used by Forensic Virtual Machines, when scanning for malicious symptoms, that arise due to malware infections. To evaluate the proposed approach, a framework, named the *Mobility Algorithm Analyser* was developed. The framework was used to simulate the process of FVMs scanning and calculate the cost results from using each mobility algorithm, based on symptoms derived from 4 malware families and 3 malware types. The results showed that there was a considerable difference in the scanning cost when using different mobility algorithms, which are embedded on each FVM to schedule the movement of FVMs from one VM to another. Using the proposed probabilistic approach, we were able to determine the most beneficial mobility algorithm, given a number of parameters that represent the status of the scanned environment; such as the number of VMs, the deployed FVMs and the cost of false positives and false negatives.

The approach proposed in this chapter could help in identifying the most beneficial mobility algorithm prior to the scan and the initialisation of FVMs, thus reducing the cost of FVM's creation, which is considered as a resource-intensive task.

Question 3: How can VMI-based lightweight monitoring approaches (e.g. FVMs) be integrated in a cloud-based malware detection system to develop efficient and effective detection of infections?

In Chapter 6 we proposed a market-inspired prioritisation approach which combined two layers of protection (i.e. signature-based and lightweight behavioural scanners) to prioritise the full malware scanning of VMs given information gathered by the small lightweight scanners. The light inspection was used to decide when and which VM should be thoroughly scanned by an AV solution; it will then trigger the full malware scan on a pre-defined percentage of the most critical VMs to accurately determine their infection status. A simulation was used to evaluate the approach; we found that

the proposed prioritisation approach helped to balance the trade-off appropriately, by confirming the infection status of these suspicious VMs, while promoting lower consumption of the cloud's VMs resources.

The approach proposed in this chapter could help in reducing the cost and resources resulting from both, flagging a machine wrongly as infected, which can incur a considerable time and effort; and using costly AV solutions to scan all the target VMs. The fact that the proposed approach combined two different malware detection methods (i.e. signature-based and behavioural-based), could offer a more effective identification of infections, as such methods are expected to complement each other. Such an integrative protection is also expected to have considerable benefit from another perspective, where VMs are protected using in-VM AV solutions and out-of-the-guest monitoring approaches, thus protecting from false conclusions that might be derived when using compromised AV solutions.

7.2 Future Work

This section discusses possible future research directions arising from Chapters 4, 5 and 6; it also identifies open questions that have emerged from this thesis.

1. **Further possible extensions to the contributions presented:**

   - **Chapter 4:** in this chapter we have provided an analysis of the causes of malware misclassification when using machine learning-based malware detection systems that are built, based on some of the widely used ML algorithms in the area of malware detection (as seen in Chapter 4, p. 67). The analysis shows that some malware samples did not behave as expected which might lead to the misclassification of those malware instances. Another possibility that can lead to malware samples being misclassified, which has not been considered in this chapter, is the choice of the classifier used, especially in the case of using

black box machine learning algorithms, such as SVM classifiers. This is due to the fact that a high classification rate might not necessarily mean that the class was predicted based on meaningful features [140]. Therefore, a possible research direction could involve conducting a thorough analysis using some recent advanced tools such as LIME [140, 139]. Such a tool can provide an additional and comprehensive interpretation of the classifier performance and results. It could also provide more guidance on whether the classifier was able to predict the true class based on reasonable features.

- **Chapter 5:** this chapter provided a probabilistic approach to analyse the performance of the mobility algorithms embedded on each FVM scanner, and nominated the most beneficial one. A number of extensions could be developed by taking into consideration cases that have not been covered in that chapter. Examples include, considering the case when a system state changes from infected to non-infected or vice versa during a scan, and the case of having multiple malware infections at the same time. The approach presented could also be evaluated further using probabilistic model checking techniques.

- **Chapter 6:** in this chapter we showed that the proposed market-based approach could be guided to scan a set of VMs which are considered more critical in terms of their infection status. This approach results in the most efficient use of the lightweight scanners, where less attention is paid to the economic value of each of the scanned VMs. This is based on the assumption that the malware infection could jeopardise the underlying cloud infrastructure; therefore, the focus was on identifying accurately those infected VMs. However, if the purpose is to give more weight to the protection of those VMs that are recognised as more economically important, the heavy-weight scanning resources could be guided to scan those VMs by reducing the value of λ in Equation 6.1 (p. 108). It would then be interesting to evaluate the decisions provided by the proposed approach from an economic perspective, and investigate how such a perspective might influence the

security decisions made. The evaluation in this case could be achieved through the use of a case study-based evaluation method; it requires a detailed analysis of the environment in question and the expected attack scenarios. Based on that, a trade-off analysis between the malware scanning cost and attack cost will be the output of such an evaluation.

2. Using dynamically extracted features with machine-learning techniques to distinguish malware from benign samples when ostensibly similar functionality is shared:

Machine learning-based malware detectors show promising results when used to distinguish malware from benign samples (as seen previously in Chapters 2 and 4). However, a number of open questions are worth investigating, especially with the current change of malware distribution where Ransomeware became the most dominant malware type. One possible direction is checking whether we could use machine learning methods with features generated using dynamic analysis to distinguish benign from malware samples when ostensibly similar functionality and behaviour is shared (such as in the case of legitimate encryption tools and Ransomware). As the behaviour generated by both of them is expected to be more or less the same (a large number of files are accessed simultaneously as the encryption process starts), the only clear difference is in the possession of the encryption keys. A classifier (such as the one presented in Chapter 4, p. 67) could be used to classify the samples and an analysis of the misclassified instances could be performed to determine whether the dynamic features generated would be sufficient, or static features might reveal more in such a case.

7.3 Closing Statement

This thesis has contributed to the development of a better understanding of malware behaviour and presented several improvements in the field of behaviour-based malware

detection. In summary, we have made three main contributions in this thesis: (i) identifying and analysing reasons that lead to malware samples being misclassified; (ii) optimising the scanning performance of Forensic Virtual Machines, and identifying the most beneficial mobility algorithm that could be used prior to the creation of FVMs; (iii) balancing the trade-off between detection accuracy and consumption of VMs' resources when using lightweight monitoring techniques, which are based on Virtual Machine Introspection technology. We believe that the improvements proposed in this thesis could help to improve malware detection by: 1. helping other researchers who intend to propose an ML-based detection system by interpreting the resulting misclassification rate, therefore, developing more robust ML-based detection systems; 2. optimising the performance of identifying malware according to their respective symptoms, by choosing the most beneficial distributed algorithm. 3. improving the detection of malware in the cloud environment through increasing the efficiency and effectiveness of mechanisms based on VMI technology.

LIST OF REFERENCES

[1] V. systems. volatility. https://www.volatilesystems.com, 2015.

[2] Zahraa S Abdallah, Mohamed Medhat Gaber, Bala Srinivasan, and Shonali Krishnaswamy. Anynovel: detection of novel concepts in evolving data streams. *Evolving Systems*, 7(2):73–93, 2016.

[3] Shahid Alam, Ibrahim Sogukpinar, Issa Traore, and Yvonne Coady. In-cloud malware analysis and detection: State of the art. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 473. ACM, 2014.

[4] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pages 171–182. Australian Computer Society, Inc., 2011.

[5] Manoun Alazab, Robert Layton, Sitalakshmi Venkataraman, and Paul Watters. Malware detection based on structural and behavioural features of api calls. 2010.

[6] Nada Alruhaily, Behzad Bordbar, and Tom Chothia. Analysis of mobility algorithms for forensic virtual machine based malware detection. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 766–773. IEEE, 2015.

[7] Nada Alruhaily, Behzad Bordbar, and Tom Chothia. Towards an understanding of the misclassification rates of machine learning-based malware detection systems. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 101–112, 2017. ISBN 978-989-758-209-7.

[8] Nada Alruhaily, Carlos Mera-Gómez, Tom Chothia, and Rami Bahsoon. A market-based approach for detecting malware in the cloud via introspection. In *International Conference on Service-Oriented Computing*, pages 722–730. Springer, 2017.

[9] Sultan S Alshamrani, Dariusz R Kowalski, and Leszek A Gasieniec. Efficient discovery of malicious symptoms in clouds via monitoring virtual machines. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 1703–1710. IEEE, 2015.

[10] Anubis. Submit to anubis script. `http://analysis.iseclab.org/Resources/submit_to_anubis.py`, 2009.

[11] Hyun Wook Baek, A. Srivastava, and J. van der Merwe. Cloudvmi: Virtual machine introspection as a cloud service. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 153–158, March 2014. doi: 10.1109/IC2E.2014.82.

[12] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In *Recent Advances in Intrusion Detection*, pages 178–197. Springer, 2007.

[13] Jamie Barnett. June 2016 - worldwide cloud report, 2017. URL `https://resources.netskope.com/h/i/262738806-june-2016-worldwide-cloud-report`.

[14] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. *TTAnalyze: A tool for analyzing malware.* 2006.

[15] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.

[16] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel,

and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer, 2009.

[17] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *USENIX workshop on large-scale exploits and emergent threats (LEET)*, 2009.

[18] Mohammad Bazm, Rida Khatoun, Youcef Begriche, Lyes Khoukhi, Xiuzhen Chen, and Ahmed Serhrouchni. Malicious virtual machines detection through a clustering approach. In *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*, pages 1–8. IEEE, 2015.

[19] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[20] C. Benninger, S.W. Neville, Y.O. Yazir, C. Matthews, and Y. Coady. Maitland: Lighter-weight vm introspection to support cyber-security in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 471–478, June 2012. doi: 10.1109/CLOUD.2012.145.

[21] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 31–38. IEEE, 2010.

[22] Bitdefender. Hypervisor introspection. `https://www.bitdefender.com/business/hypervisor-introspection.html`, June 17 2013.

[23] Behzad Bordbar and Philip Weber. Automated prevention of failure in complex and large systems: Fighting fire with fire. *Informatics Society*, page 97.

[24] Sunanda Bose, Atrayee Gupta, Sriyanjana Adhikary, and Nandini Mukherjee. Towards a sensor-cloud infrastructure with sensor virtualization. In *Proceedings of the Second Workshop on Mobile Sensing, Computing and Communication*, pages 25–30. ACM, 2015.

[25] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[26] Farid Bourennani, Ken Q Pu, and Ying Zhu. Visualization and integration of databases using self-organizing map. In *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA'09. First International Conference on*, pages 155–160. IEEE, 2009.

[27] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[28] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[29] Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. Large scale malware collection: lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*. Citeseer, 2008.

[30] CARO. Computer antivirus research organization. `http://www.caro.org/index.html`, 2001.

[31] Harlan Carvey. The windows registry as a forensic resource. *Digital Investigation*, 2(3):201–205, 2005.

[32] Ralph Cassady. *Auctions and auctioneering*. Univ of California Press, 1967.

[33] João Marcelo Ceron, Cíntia Borges Margi, and Lisandro Zambenedetti Granville. Mars: An sdn-based malware analysis solution. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 525–530. IEEE, 2016.

[34] Edward Y Chang, Beitao Li, Gang Wu, and Kingshy Goh. Statistical learning for effective visual information retrieval. In *ICIP (3)*, pages 609–612. Citeseer, 2003.

[35] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[36] M Christiansen. Bypassing malware defenses. *SANS Institute InfoSec Reading Room*, pages 3–4, 2010.

[37] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. Technical report, DTIC Document, 2006.

[38] Scott H Clearwater. *Market-based control: A paradigm for distributed resource allocation*. World Scientific, 1996.

[39] Dave Cliff and Janet Bruten. *Simple bargaining agents for decentralized market-based control*. Hewlett-Packard Laboratories, 1998.

[40] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[41] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[42] CrowdStrike. Venom: Virtualised environment neglected operations manipulation. `http://venom.crowdstrike.com`, June 17 2015.

[43] Cuckoo Sandbox. Automated malware analysis - cuckoo sandbox. `http://www.cuckoosandbox.org/`, 2015.

[44] Jingsong Cui, Hao Xiang, Chi Guo, and Kun Hou. Agentless processes monitoring architecture on cloud platform. In *Cloud Computing and Big Data (CCBD), 2014 International Conference on*, pages 1–7. IEEE, 2014.

[45] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[46] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62. ACM, 2008.

[47] Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jonathon Giffin, and Wenke Lee. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 297–312. IEEE, 2011.

[48] Shaun Donaldson, Andrei Florescu, Kurt Roemer, and Martin Zugec. Secure browsing powered by citrix xenapp, citrix xenserver direct inspect apis and bitdefender hvi. White paper, Bitdefender. URL `https://download.bitdefender.com/resources/media/materials/hypervisor-introspection/en/Bitdefender-Business-2015-WhitePaper_SecRemBro-Citrix-Crea894-en_EN-screen.pdf`.

[49] Juan Du, Xiaohui Gu, and Douglas S Reeves. Highly available component sharing in large-scale multi-tenant cloud systems. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 85–94. ACM, 2010.

[50] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6, 2012.

[51] Wedad Elmaghraby and Pinar Keskinocak. Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions.

Management Science, 49(10):1287–1309, 2003. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/4134007.

[52] Mojtaba Eskandari and Sattar Hashemi. A graph mining approach for detecting unknown malwares. *Journal of Visual Languages & Computing*, 23(3):154–162, 2012.

[53] Chun-I Fan, Han-Wei Hsiao, Chun-Han Chou, and Yi-Fan Tseng. Malware detection systems based on api log data mining. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 3, pages 255–260. IEEE, 2015.

[54] Parvez Faruki, Vijay Laxmi, Manoj Singh Gaur, and P Vinod. Behavioural detection with api call-grams to identify malicious pe files. In *Proceedings of the First International Conference on Security of Internet of Things*, pages 85–91. ACM, 2012.

[55] César Ferri, José Hernández-Orallo, and R Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009.

[56] Ivan Firdausi, Charles Lim, Alva Erwin, and Anto Satriyo Nugroho. Analysis of machine learning techniques used in behavior-based malware detection. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, pages 201–203. IEEE, 2010.

[57] Andreas Fischer, Thomas Kittel, Bojan Kolosnjaji, Tamas K Lengyel, Waseem Mandarawi, Hermann de Meer, Tilo Müller, Mykola Protsenko, Hans P Reiser, Benjamin Taubmann, et al. Cloudidea: A malware defense architecture for cloud data centers. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 594–611. Springer, 2015.

[58] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[59] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156, 1996.

[60] Akinori Fujino, Junichi Murakami, and Tatsuya Mori. Discovering similar malware samples using api call topics. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 140–147. IEEE, 2015.

[61] Yoshiro Fukushima, Akihiro Sakai, Yoshiaki Hori, and Kouichi Sakurai. A behavior based malware detection scheme for avoiding false positive. In *Secure Network Protocols (NPSec), 2010 6th IEEE Workshop on*, pages 79–84. IEEE, 2010.

[62] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, 2014, 2014.

[63] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, 2003.

[64] Saurabh Kumar Garg and Rajkumar Buyya. Green cloud computing and environmental sustainability. *Harnessing Green IT: Principles and Practices*, pages 315–340, 2012.

[65] Dragos Gavrilut, Mihai Cimpoesu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *Computer Science and Information Technology, 2009. IMCSIT'09. International Multiconference on*, pages 735–741. IEEE, 2009.

[66] MA Gibney, Nicholas R Jennings, NJ Vriend, and José-Marie Griffiths. Market-based call routing in telecommunications networks using adaptive pricing and real bidding. In *International Workshop on Intelligent Agents for Telecommunication Applications*, pages 46–61. Springer, 1999.

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

[68] Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. An approach for detection and family classification of malware based on behavioral analysis. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–5. IEEE, 2016.

[69] Keith Harrison, Behzad Bordbar, Syed TT Ali, Chris I Dalton, and Andrew Norman. A framework for detecting malware in cloud by identifying symptoms. In *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*, pages 164–172. IEEE, 2012.

[70] Safaa Salam Hatem, Mahmoud M El-Khouly, et al. Malware detection in cloud computing. *Int J Adv Comput Sci Appl*, 5(4), 2014.

[71] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[72] Jozsef Hegedus, Yoan Miche, Alexander Ilin, and Amaury Lendasse. Methodology for behavioral-based malware analysis and detection using random projections and k-nearest neighbors classifiers. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, pages 1016–1023. IEEE, 2011.

[73] Stefan Helmreich. Flexible infections: computer viruses, human bodies, nation-states, evolutionary capitalism. *Science, Technology, & Human Values*, 25(4):472–491, 2000.

[74] P Hoffman, K Scarfone, and M Souppaya. Guide to security for full virtualization technologies. *National Institute of Standards and Technology (NIST)*, pages 800–125, 2011.

[75] Tang Hongwei, Feng Shengzhong, Zhao Xiaofang, and Jin Yan. Virtav: An agentless antivirus system based on in-memory signature scanning for virtual machine. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, pages 1–2. IEEE, 2016.

[76] Fu-Hau Hsu, Min-Hao Wu, Chang-Kuo Tso, Chi-Hsien Hsu, and Chieh-Wen Chen. Antivirus software shield against antivirus terminators. *IEEE Transactions on Information Forensics and Security*, 7(5):1439–1447, 2012.

[77] Jin Huang and Charles X Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.

[78] Tongwook Hwang, Youngsang Shin, Kyungho Son, and Haeryong Park. Design of a hypervisor-based rootkit detection method for virtualized systems in cloud computing environments. In *Proceedings of the 2013 AASRI Winter International Conference on Engineering and Technology*, pages 27–32, 2013.

[79] Amani S Ibrahim, James Hamlyn-Harris, John Grundy, and Mohamed Almorsy. Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 113–120. IEEE, 2011.

[80] Rafiqul Islam, Ronghua Tian, Veelasha Moonsamy, and Lynn Batten. A comparison of the classification of disparate malware collected in different time periods. *Journal of networks*, 7(6):946–955, 2012.

[81] Rafiqul Islam, Irfan Altas, and Md Saiful Islam. Exploring timeline-based malware classification. In *IFIP International Information Security Conference*, pages 1–13. Springer, 2013.

[82] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware:

from a survey towards an established taxonomy. *Journal in computer Virology*, 4 (3):251–266, 2008.

[83] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 128–138. ACM, 2007.

[84] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC, 2017. USENIX Association. ISBN 978-1-931971-40-9. URL `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney`.

[85] Pilsung Kang and Sungzoon Cho. Eus svms: Ensemble of under-sampled svms for data imbalance problems. In *International Conference on Neural Information Processing*, pages 837–846. Springer, 2006.

[86] Masoudeh Keshavarzi. Traditional host based intrusion detection systems' challenges in cloud computing. *Advances in Computer Science: an International Journal*, 3(2):133–138, 2014.

[87] Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):552–568, 2011.

[88] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, 11(6):659101, 2015.

[89] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *USENIX Security Symposium*, pages 287–301, 2014.

[90] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Learning for Text Categorization*, pages 33–40, 1998.

[91] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[92] Kenichi Kourai and Shigeru Chiba. Hyperspector: virtual distributed monitoring environments for secure intrusion detection. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 197–207. ACM, 2005.

[93] Michal Kruczkowski and Ewa Niewiadomska Szynkiewicz. Support vector machine for malware analysis and classification. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02*, pages 415–420. IEEE Computer Society, 2014.

[94] Anthony M Kwasnica and Katerina Sherstyuk. Multiunit auctions. *Journal of economic surveys*, 27(3):461–490, 2013.

[95] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[96] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[97] Peng Li, Limin Liu, Debin Gao, and Michael K Reiter. On challenges in evaluating

malware clustering. In *Recent Advances in Intrusion Detection*, pages 238–255. Springer, 2010.

[98] LibVMI. Virtual machine introspection. `http://libvmi.com`, 2015.

[99] Wei-Jiun Lin and James J Chen. Class-imbalanced classifiers for high-dimensional data. *Briefings in bioinformatics*, page bbs006, 2012.

[100] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[101] Yi-Bin Lu, Shu-Chang Din, Chao-Fu Zheng, and Bai-Jian Gao. Using multi-feature and classifier ensembles to improve malware detection. *Journal of CCIT*, 39(2): 57–72, 2010.

[102] Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

[103] Kyle Maxwell. Mwcrawler. `https://github.com/0day1day/mwcrawler`, 2012.

[104] Kyle Maxwell. Maltrieve. `https://github.com/technoskald/maltrieve`, 2015.

[105] Vivienne Mee, Theodore Tryfonas, and Iain Sutherland. The windows registry as a forensic artefact: Illustrating evidence collection for internet usage. *Digital Investigation*, 3(3):166–173, 2006.

[106] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[107] Qiguang Miao, Jiachen Liu, Ying Cao, and Jianfeng Song. Malware detection using bilayer behavior abstraction and improved one-class support vector machines. *International Journal of Information Security*, pages 1–19, 2015.

[108] Microsoft. Trojandownloader:win32/moure.e, detected as: Trojan.fakeav (symantec). https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader%3aWin32%2fMoure.E, June 17 2013.

[109] Microsoft. Malware encyclopedia. http://www.microsoft.com/security/portal/threat/Threats.aspx, 2015.

[110] Microsoft. Microsoft security intelligence report (sir). http://www.microsoft.com/security/sir/default.aspx, 2015.

[111] Microsoft Azure. Vm agent and extensions. https://azure.microsoft.com/en-us/blog/vm-agent-and-extensions-part-2/t, April 2014.

[112] Pascal Millaire. Symantec: Wannacry ransomware: 6 implications for the insurance industry. https://www.symantec.com/connect/blogs/wannacry-ransomware, May 15 2017.

[113] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN 9780071154673. URL https://books.google.co.uk/books?id=EoYBngEACAAJ.

[114] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring multiple execution paths for malware analysis. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 231–245. IEEE, 2007.

[115] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE, 2007.

[116] Robert Moskovitch, Clint Feher, and Yuval Elovici. Unknown malcode detectiona chronological evaluation. In *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on*, pages 267–268. IEEE, 2008.

[117] Ashwini Mujumdar, Gayatri Masiwal, and Dr BB Meshram. Analysis of signature-based and behavior-based anti-malware approaches. *signature*, 2(6), 2013.

[118] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[119] Vivek Nallur. *A decentralized self-adaptation mechanism for service-based applications in the cloud*. PhD thesis, University of Birmingham, 2012.

[120] Kara Nance, Matt Bishop, and Brian Hay. Virtual machine introspection: Observation or interference? *IEEE Security & Privacy*, 6(5), 2008.

[121] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. Context-aware, adaptive and scalable android malware detection through online learning (extended version). *arXiv preprint arXiv:1706.00947*, 2017.

[122] Julia Narvaez, Barbara Endicott-Popovsky, Christian Seifert, Chiraag Aval, and Deborah A Frincke. Drive-by-downloads. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[123] Pratiksha Natani and Deepti Vidyarthi. Malware detection using api function frequency with ensemble based classifier. In *International Symposium on Security in Computing and Communication*, pages 378–388. Springer, 2013.

[124] Norman Sandbox. Norman safeground antivirus software. `http://www.norman.com/index/`.

[125] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Rethinking antivirus: Executable analysis in the network cloud. In *HotSec*, 2007.

[126] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloudav: N-version antivirus in the network cloud. In *USENIX Security Symposium*, pages 91–106, 2008.

[127] Offensivecomputing. Open malware. `http://www.offensivecomputing.net`, 2015.

[128] Anthony O'Hagan and Jonathan J Forster. *Kendall's advanced theory of statistics, volume 2B: Bayesian inference*, volume 2. Arnold, 2004.

[129] OpenCV. Introduction to support vector machines. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html.

[130] Martin J Osborne and Ariel Rubinstein. *Bargaining and markets*. Academic press, 1990.

[131] Panda Labs. Panda labs q1 report: Trojans account for 80% of malware infections, set new record. https://www.pandasecurity.com/mediacenter/press-releases/pandalabs-q1-report-trojans-account-for-80-of-malware-infections-set-new-record/, May 2013.

[132] Panda Labs. 20% of all malware ever created appeared in 2013. https://www.pandasecurity.com/mediacenter/press-releases/20-of-all-malware-ever-created-appeared-in-2013/, March 2014.

[133] Bryan D Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 233–247. IEEE, 2008.

[134] Naser Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 300–305. IEEE, 2013.

[135] Abdurrahman Pektaş, Tankut Acarman, Yliès Falcone, and Jean-Claude Fernandez. Runtime-behavior based malware classification using online machine learning. In *2015 World Congress on Internet Security (WorldCIS)*, pages 166–171. IEEE, 2015.

[136] Radu S Pirscoveanu, Steven S Hansen, Thor MT Larsen, Matija Stevanovic, Jens Myrup Pedersen, and Alexandre Czech. Analysis of malware behavior: Type

classification using machine learning. In *Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2015 International Conference on*, pages 1–7. IEEE, 2015.

[137] Yong Qiao, Jie He, Yuexiang Yang, and Lin Ji. Analyzing malware by abstracting the frequent itemsets in api call sequences. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 265–270. IEEE, 2013.

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

[139] Marco Tulio Ribeiro. Lime. `https://github.com/marcotcr/lime`, 2016.

[140] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[141] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[142] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[143] Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. Using feature generation from api calls for malware detection. *Computer Fraud & Security*, 2014(9):9–18, 2014.

[144] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 1020–1025. ACM, 2010.

[145] Saudi Aramco. Saudi arabian oil co. http://www.saudiaramco.com/en/home.html, December 2016.

[146] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2): 197–227, 1990.

[147] Shane Schick. Security intelligence: Tinba malware watches mouse movements, screen activity to avoid sandbox detection. https://securityintelligence.com/news/tinba-malware-watches-mouse-movements-screen-activity-to-avoid-sandbox-detection/, 2016.

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

[149] Scikit-learn. Scikit-learn: machine learning in python. http://scikit-learn.org/stable/, June 17 2013.

[150] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[151] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev, and Yuval Elovici. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, 1(1):1–22, 2012.

[152] Claude E Shannon and Warren Weaver. The mathematical theory of information. 1949.

[153] Monirul I Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. Secure in-vm monitoring using hardware virtualization. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 477–487. ACM, 2009.

[154] Adrian L Shaw, Behzad Bordbar, John Saxon, Keith Harrison, and Chris I Dalton. Forensic virtual machines: dynamic defence in the cloud via introspection. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 303–310. IEEE, 2014.

[155] Adrian L Shaw, Behzad Bordbar, John Saxon, Keith Harrison, and Chris I Dalton. Forensic virtual machines: dynamic defence in the cloud via introspection. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 303–310. IEEE, 2014.

[156] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 81–92. ACM, 2012.

[157] Kuyoro So. Cloud computing security issues and challenges. *International Journal of Computer Networks*, 3(5):247–55, 2011.

[158] Sahil Suneja, Canturk Isci, Eyal de Lara, and Vasanth Bala. Exploring vm introspection: Techniques and trade-offs. In *ACM SIGPLAN Notices*, volume 50, pages 133–146. ACM, 2015.

[159] Symantec. Understanding heuristics - symantecs bloodhound technology. `https://www.virusbulletin.com/uploads/pdf/magazine/1996/199611.pdf`, November 1996.

[160] Symantec. Infostealer. `https://www.symantec.com/security_response/writeup.jsp?docid=2000-122016-0558-99&tabid=2`, February 15 2013.

[161] Symantec. W32.xpiro.i. `https://www.symantec.com/security_response/writeup.jsp?docid=2015-102911-0452-99`, June 17 2013.

[162] Symantec. Symantec security response - virus naming conventions. `https://www.symantec.com/security_response/virusnaming.jsp`, June 17 2013.

[163] Symantec. Adware.popuppers. https://www.symantec.com/security_response/writeup.jsp?docid=2005-022816-5036-99&tabid=2, June 17 2013.

[164] Symantec. W32.sality.ae. https://www.virustotal.com/en/file/bbc345fb099248fad14e8d012c3bc11e16ea27a5f291f13bc79c181913e395d9/analysis/, June 17 2013.

[165] Symantec. W32.sality!dam. https://www.symantec.com/security_response/writeup.jsp?docid=2013-043010-4816-99, June 17 2013.

[166] Symantec. Internet security threat report. http://www.symantec.com/security_response/publications/threatreport.jsp, 2015.

[167] Symantec. Trojan.gen. https://www.symantec.com/security_response/writeup.jsp?docid=2010-022501-5526-99, 2016.

[168] Symantec. Internet security threat report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf, April 2016.

[169] Symantec. A-z listing of threats & risks. https://www.symantec.com/security_response/landing/azlisting.jsp, 2016.

[170] Symantec Security Response. The shamoon attacks. https://www.symantec.com/connect/blogs/shamoon-attacks, August 16 2012.

[171] Symantec Security Response. Shamoon: Back from the dead and destructive as ever. https://www.symantec.com/connect/blogs/shamoon-back-dead-and-destructive-ever, November 30 2016.

[172] Symantec Security Response. Latest intelligence. https://www.symantec.com/security_response/publications/monthlythreatreport.jsp, July 05 2017.

[173] Peter Szor. *The art of computer virus research and defense*. Pearson Education, 2005.

[174] Hongwei Tang, Shengzhong Feng, Xiaofang Zhao, and Yan Jin. Virtav: An agentless antivirus system based on in-memory signature scanning for virtual machine. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, pages 124–133. IEEE, 2016.

[175] Threat Expert. Automated threat analysis. `http://www.threatexpert.com/`.

[176] Ronghua Tian, Rafiqul Islam, Lynn Batten, and Steve Versteeg. Differentiating malware from cleanware using behavioural analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 23–30. IEEE, 2010.

[177] Fotis Tsifountidis. Virtualization security: Virtual machine monitoring and introspection. *Signature*, 2010.

[178] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[179] Paul Tucker. Market mechanisms in a programmed system. *Department of Computer Science and Engineering, University of California*, 130, 1998.

[180] Xabier Ugarte-Pedrero, Igor Santos, Carlos Laorden, Borja Sanz, and Pablo G Bringas. Collective classification for packed executable identification. *International Journal of Computer Systems Science & Engineering*, 28(1):25–36, 2013.

[181] R Veeramani and Nitin Rai. Windows api based malware detection and framework analysis. In *International conference on networks and cyber security*, volume 25, 2012.

[182] Virus Bulletin. The international publication on computer virus prevention, recognition and removal. `https://www.virusbulletin.com/uploads/pdf/magazine/1996/199611.pdf`, November 1996.

[183] Virusshare. Virusshare.com. `http://vxheaven.org`, 2016.

[184] VirusTotal. Free online virus, malware and url scanner. https://www.virustotal.com/, 2015.

[185] VMware. Enabling vmware vshield endpoint in a vmware horizon view environment. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-horizon-view-vshield-endpoint-antivirus-white-paper.pdf, October 2015.

[186] VX Heaven. Vxheaven.org. http://vxheaven.org, 2016.

[187] Andrew Walenstein and Arun Lakhotia. The software similarity problem in malware analysis. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[188] Cheng Wang, Jianmin Pang, Rongcai Zhao, and Xiaoxian Liu. Using api sequence and bayes algorithm to detect suspicious behavior. In *Communication Software and Networks, 2009. ICCSN'09. International Conference on*, pages 544–548. IEEE, 2009.

[189] Cheng Wang, Zheng Qin, Jixin Zhang, and Hui Yin. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on*, pages 481–487. IEEE, 2016.

[190] Gang Wang, Jinxing Hao, Jian Ma, and Hongbing Jiang. A comparative assessment of ensemble learning for credit scoring. *Expert systems with applications*, 38(1):223–230, 2011.

[191] Xiaodong Wang and José F Martínez. Xchange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 113–125. IEEE, 2015.

[192] Florian Westphal, Stefan Axelsson, Christian Neuhaus, and Andreas Polze. Vmi-pl: A monitoring language for virtual platforms using virtual machine introspection. *Digital Investigation*, 11:S85–S94, 2014.

[193] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[194] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2):32–39, 2007.

[195] David E Williams. *Virtualization with Xen (tm): Including XenEnterprise, XenServer, and XenExpress: Including XenEnterprise, XenServer, and XenExpress*. Syngress, 2007.

[196] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[197] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[198] Christian Wressnegger, Kevin Freeman, Fabian Yamaguchi, and Konrad Rieck. From malware signatures to anti-virus assisted attacks. *arXiv preprint arXiv:1610.06022*, 2016.

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

[200] J-Y Xu, Andrew H Sung, Patrick Chavez, and Srinivas Mukkamala. Polymorphic malicious executable scanner by api sequence analysis. In *Hybrid Intelligent Systems, 2004. HIS'04. Fourth International Conference on*, pages 378–383. IEEE, 2004.

[201] Wei Yan, Zheng Zhang, and Nirwan Ansari. Revealing packed malware. *ieee seCurity & PrivaCy*, 6(5), 2008.

[202] Bee Wah Yap, Khatijahhusna Abd Rani, Hezlin Aryani Abd Rahman, Simon Fong, Zuraida Khairudin, and Nik Nik Abdullah. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013)*, pages 13–22. Springer, 2014.

[203] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang. An intelligent pe-malware detection system based on association mining. *Journal in computer virology*, 4(4):323–334, 2008.

[204] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5(4):283–293, 2009.

[205] Yanfang Ye, Tao Li, Kai Huang, Qingshan Jiang, and Yong Chen. Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list. *Journal of Intelligent Information Systems*, 35(1):1–20, 2010.

[206] Bo-yun Zhang, Jian-ping Yin, Jin-bo Hao, Ding-xing Zhang, and Shu-lin Wang. Using support vector machine to detect unknown computer viruses. *International Journal of Computational Intelligence Research*, 2(1):100–104, 2006.

[207] Boyun Zhang, Jianping Yin, Wensheng Tang, Jinbo Hao, and Dingxing Zhang. Unknown malicious codes detection based on rough set theory and support vector machine. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 2583–2587. IEEE, 2006.

[208] Yi Zhang and Xiaoming Jin. An automatic construction and organization strategy for ensemble learning on data streams. *ACM SIGMOD Record*, 35(3):28–33, 2006.

[209] Hengli Zhao, Ming Xu, Ning Zheng, Jingjing Yao, and Qiang Ho. Malicious executables classification based on behavioral factor analysis. In *e-Education, e-Business,*

e-Management, and e-Learning, 2010. IC4E'10. International Conference on, pages 502–506. IEEE, 2010.

[210] Mikhail Zolotukhin and Timo Hamalainen. Detection of zero-day malware based on the analysis of opcode sequences. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 386–391. IEEE, 2014.