



# **Malware Analysis and Detection in Enterprise Systems**

**Tebogo Mokoena**

**Student Number: 20255039**

Dissertation submitted in fulfilment of the requirements for the degree Magister Technologiae in  
Information Technology, Faculty of Applied and Computer Sciences,  
Vaal University of Technology

**Supervisor: Professor Tranos Zuva**  
**Co-Supervisor: Anneke Harmse**

**March 2017**

**© Copy Right by Mokoena T**  
**November 2016**  
**All rights reserved**

## DECLARATION OF AUTHORSHIP

I, Tebogo Mokoena, hereby declare that the dissertation entitled Malware Analysis and Detection in Enterprise Systems submitted here, is the product of my own independent research.

All the sources I have used and quoted have been pointed out and acknowledged by means of complete references.

In addition, I declare that the work is submitted for the first time at this university / faculty towards the Magister Technologiae degree in the Information Technology department and that it has never been submitted to any other university / faculty for the purpose of obtaining a degree.

---

Signature

---

Date

## **ACKNOWLEDGEMENTS**

During the course of this research, I was privileged to work with, and enjoy the support of my supervisor, Professor Tranos Zuva without whose knowledge and guidance this project would never have reached completion.

I am also deeply indebted to my colleagues who assisted in reviewing this dissertation.

## **DEDICATION**

This dissertation is dedicated to my late grandmother, my parents and my beloved wife Sandy, who supported and encouraged me to accomplish this work throughout her pregnancy and blessed me with our baby boy Khanya.

## **ABSTRACT**

Malware is today one of the biggest security threats to the Internet. Malware is any malicious software with the intent to perform malevolent activities on a targeted system. Viruses, worms, trojans, backdoors and adware are but a few examples that fall under the umbrella of malware.

The purpose of this research is to investigate techniques that are used in order to effectively perform Malware analysis and detection on enterprise systems to reduce the damage of malware attacks on the operation of organizations.

Malware analysis experiments were carried out using the two techniques of malware analysis, which are Dynamic and Static analysis, on two different malware samples. Portable executable and Microsoft word document files were the two samples that were analysed in an isolated sandbox lab environment.

Static analysis is the process of examining and extracting information from malware code without executing the malware, while Dynamic analysis is the process of executing malware in order to observe and record its behaviour in a controlled environment.

The results from the experiments disclosed the behaviour, encryption techniques, and other techniques employed by the malware samples. These malware analysis experiments were carried out in an isolated lab environment that was built for the purpose of this research.

The results showed that Dynamic analysis is more effective than Static analysis. The study proposes the use of both techniques for comprehensive malware analysis and detection.

# TABLE OF CONTENTS

<b>DECLARATION OF AUTHORSHIP</b> .....	i
<b>ACKNOWLEDGEMENTS</b> .....	ii
<b>DEDICATION</b> .....	iii
<b>ABSTRACT</b> .....	iv
<b>TABLE OF CONTENTS</b> .....	v
<b>ACRONYMS AND ABBREVIATIONS</b> .....	x
<b>CHAPTER 1</b> .....	1
CHAPTER OVERVIEW .....	1
1 INTRODUCTION.....	1
1.1 Problem Statement .....	5
1.2 Research Question .....	6
1.3 Objectives of Research.....	6
1.4 Aim of Research .....	6
1.5 Significance of Study .....	6
1.6 Document and Chapter Structure.....	7
1.7 Chapter Summary.....	7
<b>CHAPTER 2</b> .....	9
CHAPTER OVERVIEW .....	9
2 INTRODUCTION .....	9
2.1 Introduction to Malware.....	10
2.2 Malware and History of Malware .....	10
2.2.1 History of Malware .....	10
2.2.2 Malware Types .....	11
2.2.3 Overall Characteristics of Malware.....	15
2.2.4 Categories and Behaviour of Malware .....	16
2.2.5 Malware Functionality .....	16
2.2.6 Malware Lifecycle Model.....	20
2.3 Malware analysis .....	21
2.3.1 The goals of Malware analysis .....	23
2.3.2 Challenges of Malware Analysis .....	24
2.3.3 Limitations of Malware Analysis .....	24
2.3.4 Malware Analysis Types .....	25
2.3.5 Malware Analysis Process Overview .....	30
2.3.6 Protective Mechanisms.....	30
2.3.7 Malware Dependencies .....	30

2.4	Malware Detection Techniques .....	31
2.4.1	Signature Based Detection .....	31
2.4.2	Anomaly Based Detection.....	32
2.5	Malware Anti-reverse-engineering and anti-analysis .....	33
2.5.1	Malware anti-analysis .....	33
2.5.2	Malware anti-reverse-engineering.....	34
2.6	Malware Analysis Lab .....	50
2.6.1	Architecture Overview .....	50
2.6.2	Physical and Virtual Environments.....	52
2.6.3	Malware Analysis Isolated Environment.....	53
2.6.4	Malware Analysis System Sandboxes Overview .....	55
2.6.5	Tools based on type of analysis .....	59
2.6.6	Malware Analysis Environment Setup Overview .....	63
2.6.7	Summary of the Malware Analysis Environment Design Overview.....	65
2.7	Chapter Summary.....	66
<b>CHAPTER 3</b>	.....	<b>68</b>
RESEARCH METHODOLOGY	.....	68
CHAPTER OVERVIEW	.....	68
3 INTRODUCTION	.....	68
3.1	Research Design .....	68
3.2	A Model of the Research Process.....	70
3.2.1	Research Paradigms .....	71
3.2.2	Selected Empirical Research Method .....	76
3.3	Research Methodology .....	76
3.3.1	Selective Research Method - Quantitative Research .....	76
3.4	Data Preparation and Data Source .....	77
3.5	Data Collection Techniques .....	77
3.6	Sampling Techniques .....	79
3.7	Quantitative Data Analysis of Collected Malware .....	80
3.8	Research Methodology Actions Overview .....	81
3.9	Chapter Summary.....	82
<b>CHAPTER 4</b>	.....	<b>83</b>
MALWARE ANALYSIS EXPERIMENTS	.....	83
CHAPTER OVERVIEW	.....	83
4 INTRODUCTION	.....	83
4.1	Malware Analysis Experiments .....	83
4.1.1	Malware Analysis Process Overview .....	83
4.2	Malware Analysis Experiments .....	85
4.2.1	Analysis of Malware artifact - Ticket_354041 .....	85
4.2.2	Analysis of 1123211 – 090SD.exe .....	101



4.3 Chapter summary .....	117
<b>CHAPTER 5 .....</b>	<b>119</b>
EXPERIMENT RESULTS DISCUSSION AND CONCLUSION .....	119
CHAPTER OVERVIEW .....	119
5 INTRODUCTION .....	119
5.1 Experiment Results Discussion.....	119
5.1.1 Static Analysis Results Overview .....	119
5.1.2 Dynamic Analysis Results Overview .....	124
5.2 Measurement of Malware Analysis Techniques and detection .....	126
5.2.1 Analysis Method Results comparison .....	126
5.2.2 Effective Method Criteria Calculation .....	128
5.3 Research Summary .....	130
5.4 Research Conclusion and Recommendation .....	132
5.5 Future Research .....	135
<b>References .....</b>	<b>136</b>
Appendix A.....	143
Appendix B.....	153
Appendix E.....	154
Dynamic Analysis - Ticket_354041 .....	154
Appendix F .....	154
Dynamic Analysis - 1123211-090SD.exe.....	154
Appendix G – CD / USB Contents .....	154

## List of Figures

Figure 1: Total Malware analysed in 2015 (Paganini, 2015).....	3
Figure 2: Evolution of Malware (Touchette, 2016) .....	11
Figure 3: Malware Lifecycle (Rossow, 2013).....	20
Figure 4: Summary Malware Analysis Types .....	25
Figure 5: Overview Static Analysis Process .....	27
Figure 6: Overview Dynamic Analysis Process .....	29
Figure 7: Malware Analysis Process overview.....	30
Figure 8: A classification of malware detection techniques (Idika and Mathur, 2007). .....	32
Figure 9: View of the PE File Format (Shaban, 2013) .....	35
Figure 10: DOS-MZ Header - 1123211-090SD.exe.....	36
Figure 11: PE Section Header - 1123211-090SD.exe .....	37
Figure 12: Section Headers.....	37
Figure 13: Linker creation of executable file (Saffaf, 2009).....	38
Figure 14: Imported and Exported Functions (Saffaf, 2009) .....	39
Figure 15: Entropy Value .....	40
Figure 16: UPX Output Results (Section 4.2.2 Analysis of 1123211 – 090SD.exe) .....	42
Figure 17: PE structure of a packed executable (Pietrek, 1994).....	43
Figure 18: XOR operation-encoder .....	47
Figure 19: XOR operation – decoder.....	47
Figure 20: Virtual Box Software running Windows 7 System.....	54

Figure 21: REMnux tools overview – PESCANNER (Westcott and Zeltser, 2016) .....	56
Figure 22: REMnux tools overview – INETSIM (Westcott and Zeltser, 2016) .....	56
Figure 23: Cuckoo Sandbox Overview (Guarnieri et al., 2013).....	57
Figure 24: IRMA Web Interface Overview (Quint et al., 2016).....	58
Figure 25: Virus Total Web Interface Overview (Total, 2016) .....	59
Figure 26: REMnux Architecture Overview (Zeltser, 2016).....	64
Figure 27: IRMA Architecture Overview (Quint et al., 2016) .....	64
Figure 28: Malware Analysis Environment Overview.....	65
Figure 29: Model of the research process showing the variety of paths that can be undertaken (Brand, 2010) .....	70
Figure 30: Research Methodology Actions Overview .....	81
Figure 31: Malware Analysis Process Flowchart .....	84
Figure 32: Listing of File Section Hashes – ticket_354041.doc.....	86
Figure 33: Listing of File Properties - ticket_354041.doc.....	87
Figure 34: Listing of File Signatures - ticket_354041.doc .....	87
Figure 35: Indicators of Compromise - ticket_354041.doc.....	88
Figure 36: Blacklisted Strings - ticket_354041.doc .....	89
Figure 37: Listing of Obfuscated Code - ticket_354041.doc .....	89
Figure 38: Listing of Embedded URLs - ticket_354041.doc.....	90
Figure 39: Listing of Embedded Email Address - ticket_354041.doc.....	90
Figure 40: Listing of PE Headers - ticket_354041.doc.....	90
Figure 41: Listing of Embedded Domains - ticket_354041.doc.....	91
Figure 42: Listing of Installed Program Execution - ticket_354041.doc.....	91
Figure 43: Listing of Interesting Words Found - ticket_354041.doc .....	91
Figure 44: Listing of OLE Headers - ticket_354041.doc .....	92
Figure 45: Listing of Indication of Existence of Macros - ticket_354041.doc .....	92
Figure 46: Listing of VBA Macros Found - ticket_354041.doc .....	92
Figure 47: Listing of VBA Macros Attributes - ticket_354041.doc .....	93
Figure 48: Listing of Extracted VBA Macros - ticket_354041.doc .....	93
Figure 49: Listing of VBA Macros Code - ticket_354041.doc.....	94
Figure 50: Listing of Results Summary - ticket_354041.doc.....	95
Figure 51: Ticker_354041 Sample Overview ticket_354041.doc .....	95
Figure 52: Ticker_354041 Sample Suspicious Capabilities ticket_354041.doc .....	96
Figure 53: File Properties ticket_354041.doc .....	97
Figure 54: Activity Screenshot - ticket_354041.doc.....	97
Figure 55: File Signatures - ticket_354041.doc .....	98
Figure 56: Files dropped by the Malware - ticket_354041.doc.....	98
Figure 57: File Written - ticket_354041.doc.....	99
Figure 58: File opened - ticket_354041.doc .....	99
Figure 59: Registry Keys Read - ticket_354041.doc.....	99
Figure 60: Process Executed ticket_354041.doc.....	100
Figure 61: Results Overview .....	100
Figure 62: Virus Total Results - ticket_354041.doc .....	101
Figure 63: Listing of File Section Hashes – 1123211-090sd.exe .....	102
Figure 64: Listing of File Details - 1123211-090sd.exe.....	102
Figure 65: Listing of File Signature - 1123211-090sd.exe .....	103
Figure 66: Indicators of Compromise – 1123211-090sd.exe .....	103
Figure 67: Strings – 1123211-090sd.exe.....	104
Figure 68: Listing of String Offsets - 1123211-090sd.exe.....	104
Figure 69: Listing of Extracted Code - 1123211-090sd.exe.....	105
Figure 70: Listing of Packer PEDUMP - 1123211-090sd.exe .....	106
Figure 71: Listing of File Entropy - 1123211-090sd .....	106
Figure 72: Listing of IMPORTS PEDUMP- 1123211-090sd.exe .....	107
Figure 73: Listing of DOS STUB - 1123211-090sd.exe .....	107
Figure 74: Listing of File Directory Listing - 1123211-090sd.exe .....	108

Figure 75: Optional Headers – 1123211-090sd.exe .....	108
Figure 76: Listing of Packer PEFRAME Results- 1123211-090sd.exe .....	109
Figure 77: Listing of Suspicious Sections PEFRAME Results- 1123211-090sd.exe .....	109
Figure 78: Listing of Packer PEFRAME Results- 1123211-090sd.exe .....	109
Figure 79: Dependency Walker – 1123211-090sd.exe .....	110
Figure 80: Static Analysis Results Overview - 1123211-090SD.exe .....	110
Figure 81: File Details – 1123211-090sd.exe .....	111
Figure 82: File Signatures – 1123211-090sd.exe .....	112
Figure 83: Dropped Files – 1123211-090SD.exe .....	113
Figure 84: Hosts Involved – 1123211-090SD.exe .....	113
Figure 85: DNS Requests– 1123211-090SD.exe .....	114
Figure 86: File Read – 1123211-090SD.exe .....	114
Figure 87: File Written– 1123211-090SD.exe .....	115
Figure 88: File Deleted – 1123211-090SD.exe .....	115
Figure 89: File Opened – 1123211-090SD.exe .....	115
Figure 90: File Copied – 1123211-090SD.exe .....	115
Figure 91: Processes – 1123211-090SD.exe .....	116
Figure 92: Results Overview .....	116
Figure 93: Virus Total Results – 1123211-090SD.exe .....	117
Figure 94: Experiment 1 .....	130
Figure 95: Experiment 2 .....	130
Figure 96: Experiment 1 Analysis and Conclusion .....	133
Figure 97: Experiment 1 Analysis and Conclusion. ....	134

## List of Tables

Table 1: Summary overview of research questions and objectives.....	8
Table 2: Malware Types Summary (Uppal et al., 2004).....	15
Table 3: Sections of a PE File for a Windows Executable (Shaban, 2013, Idika and Mathur, 2007, Marak, 2015, Elisan, 2015).....	36
Table 4: Physical Machines Environment.....	52
Table 5: Virtual Machine Environment.....	52
Table 6: Toolsets .....	62
Table 7: Differences between exploratory and conclusive research .....	70
Table 8: Positivism and Interpretivism (Bajpai, 2011) .....	73
Table 9: Data Collection Techniques Used by Research Approaches (Morgan and Harmo, 2001) .....	77
Table 10: Sampling Techniques (Dudovskiy, 2011) .....	79
Table 11: Advantages and Disadvantages of Probability Sampling (Dudovskiy, 2011).....	80
Table 12: Analysis Methods results comparison.....	126
Table 13: Experiment 2 Results .....	128

## ACRONYMS AND ABBREVIATIONS

<b>APIs</b>	Application programming interfaces
<b>AV</b>	Antivirus
<b>CPU</b>	Central Processing Unit
<b>C&amp;C</b>	Command-and-control
<b>DBIR</b>	Data Breach Investigation Report
<b>DDos</b>	Distributed Denial of Service
<b>DGA</b>	Domain generation algorithms
<b>DNS</b>	Domain Name Server
<b>IOC</b>	Indicators of Compromise
<b>POS</b>	Point-of-sale
<b>PUPs</b>	Potentially undesirable processes
<b>OS</b>	Operating System
<b>OLE</b>	Object Linking and Embedding
<b>OEP</b>	Original Entry Point
<b>OCX</b>	OLE Control Extension
<b>RAM</b>	Random Access Memory
<b>VM</b>	Virtual Machine
<b>XCSB</b>	Optimising Structured BASIC compiler for microcontrollers

# CHAPTER 1

## INTRODUCTION

### CHAPTER OVERVIEW

This chapter focuses on the following areas:

- Introduction
- Problem Statement
- Research Question
- Objectives of Research
- Aim of Research
- Significance of Study
- Document Structure
- Chapter Summary

### 1 INTRODUCTION

The Internet has become an important part of people's everyday life as well as in organisations. It also plays a vital role as it allows collaboration to take place between individuals and organisations that allows them to communicate and carry out business processes. Internet connectivity provides vast benefits in relation to increased access to knowledge and information, but using the Internet can become a hazardous experience where users and organisations lack the appropriate levels of security awareness and knowledge. Anyone that uses a computer may have encountered a direct or indirect experience with malware, especially if the computer has access to the internet. Malware poses security threats to online users as the Internet was never designed to be a very secure environment.

It is important to first understand the term malware for the purpose of this research. Herewith a definition:

The term malware is short for malicious software that refers to software that is designed to cause harm or damage or even perform other unwanted actions on a computer system. Malware behaviour ranges from being an irritation such as causing advertising pop-up's, to actions that are much more harmful, such as theft of credentials and sensitive data; even infecting other machines on the network (Agrawal et al., 2014).

Malware is also designed for financial gain such as ransomware malware and while others are designed to gather sensitive information, gain access to private computer systems, or display unwanted advertising (Bhojani, 2014), some are even designed or used to accomplish civil (e.g. the Anonymous group) or politically motivated attacks. The Malware family includes trojan horses, worms, spyware, and adware. Advanced malware such as ransomware is used to commit financial fraud and extort money from computer users (Goertzel and Winograd, 2009).

Malware attacks have been around since the first well-known malicious software, which was written in the early 1950s, widely known as a “self-reproducing machine”. The evolution of malware over the years has contributed towards major security incidents or breaches that have caused major financial losses as well as reputational damage to many organizations. One of the examples of these malware attacks was the Sony Pictures hack in December 2014. According to a number of reports and security researchers, the malware that infected computer systems at Sony Pictures was named “Wiper” malware (Gallagher, 2014). The malware was involved in the vast majority of these attacks according to the Verizon’s 2016 Data Breach Investigation Report (DBIR). The DBIR report indicated that the nine major security incident classification categories were the following:

- Web Application attacks
- Point-of-sale intrusions
- Privilege misuse and insider threat
- Various errors
- Physical loss and theft
- Crimeware
- Payment card skimmers
- Cyber-espionage
- Denial of service attacks

The increase of malware attacks and the increasingly resourceful ways in which it is being used to commit crime such as to conduct espionage, steal personal information, cause destruction to governments and business operations, or deny the user access to information and services as well as defacing websites, is potentially a serious threat to the Internet economy (Privacy, 2008).

Instances of Ransomware, a class of malware, which was created solely to extort money from the intended victim or company, have seen a dramatic proliferation over the past year. The Verizon’s 2016 Data Breach Investigation Report also indicated that financial gain was far the

most common motivation for attacks and breaches, with 89% of breaches having a financial or espionage motive (Agrawal et al., 2014). Different strains of malware samples are detected on a daily basis and 12 million fresh strains of malware detected every month are just one aspect of the interesting analysis. AV-Test is an independent company that is responsible to evaluate and to rate antivirus security solutions for operating systems such as Microsoft Windows and Android operating systems. This company has revealed that every second, four samples are available in the wild.

Figure 1 below depicts the number of samples that were analysed by AV-Test in the year 2015. AV-Test analysed 18 million new samples in August 2015 and 325 million samples were analysed in December 2015 alone. The AV-Test institute registers over 390,000 new malicious programs every day (Paganini, 2015).

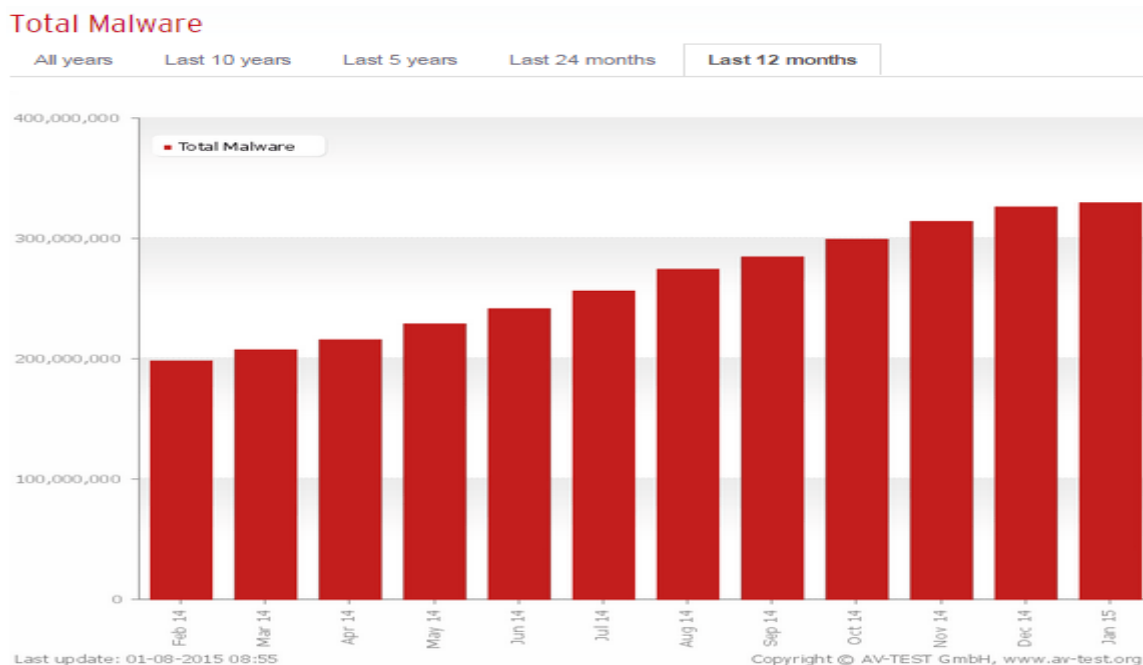


Figure 1: Total Malware analysed in 2015 (Paganini, 2015)

Malware authors are individuals, groups of hackers, national-states or even organisations (SANS610, 2012). Malware authors are designing sophisticated malware that can evade detection, making it difficult or impossible to be detected by traditional solutions such as Antivirus solutions (Mohanty, 2017). Other malware uses a variety of techniques, which include anti-sandbox or analysis detection mechanism to defeat sandbox technology, that also makes it difficult for malware analysts to analyse in a sandbox environment (Chen et al., 2008).

Malware authors have evolved sophisticated malicious code to such an extent that it is at an advanced level of complexity and uses various obfuscation methods, making analysis and detection very difficult, and in some cases virtually impossible. While obfuscation is one of the

elements of metamorphic viruses, it is usually more employed to prevent analysis and developing AV patterns. As technology has evolved, the detection methods that Antivirus vendors use have changed as well, improving their accuracy for newly-released malware. These vendors have been lacking in detecting zero-day malware instances and instances that are customised to the target and have not been seen in the wild (Pirscoveanu, 2015) or new strains of malware. The modus operandi of an Antivirus is that it uses two common methods, which are Signature and Behaviour detection, to detect and fight malware on an infected system. Signature based detection is the most common method used by antivirus solutions and vendors. This method involves searching for the detection signatures of the already classified malware within its own database that entails searching through strings of bits that are unique to the particular type of malware that is being analysed (Choudhary et al., 2013).

The behaviour method uses heuristic methods or algorithms to detect unknown malicious activities or behaviour of the malware. This method entails performing a set of tasks or actions on the malware, and the algorithm that is involved in the task analyses the patterns to identify malicious activities. The advantage of this method is that it has the ability to not only detect new malware that antivirus vendors have not yet defined in its signature database, but also may report on false positives (Choudhary et al., 2013).

Malware attack vectors include endpoints in the form of accessing an infected website, receiving a phishing email with malware attached or as the payload, and from end users who plug a malware infected device into their personal or corporate computers.

While malware can perform all sorts of malicious activities on a computer system it is then up to the malware analysts or security experts to determine the best and most efficient way in which they can respond to malware incidents or attacks. Malware analysts or security experts must have various sets of skills, and must also make use of various tools and techniques to defend against malware. As a malware analyst it is important to understand the malware analysis process, which is defined as the art of dissecting malware to understand how it works, how to identify it, and how to defeat or eliminate it (Sikorski, 2012).

We have also seen an organisation closing down or even shutting down their operations because of incidents caused by malware infections. One of the examples was the zero-day attack on Lincolnshire County Council (Burton, 2016). IT systems across the organisation were shut down due to a cyber-attack in which sensitive personal information from its adult care system was compromised. Lincolnshire County reported that the malware that infected their systems had a payload that contained zero-day malware. A zero-day event occurs when a vulnerability is newly discovered in a system and exploited. The vulnerability exploited would generally be an otherwise unknown security flaw in common applications, or malware that is



created to specifically and persistently penetrate an organisation and the malware has therefore not shown up in the wild yet.

## 1.1 Problem Statement

The prevalence of Malware is a critical security issue for organizations using the internet today. Malware is a malicious software such as viruses, botnets and Trojans, which is a problem that is continuing to evolve, and it is also the most common infection affecting computers of many users today (Distler and Hornat, 2007). It affects the system and causes a lot of operational problems. The Symantec report (Reavis, 2012) stated that in 2010 alone, 286 million different types of malware were responsible for more than 3 billion total attacks on computer users; these are staggering numbers that are just one simple measure of malware's impact.

According to the 2016 Data Breach Investigations Report, 39 percent of Crimeware and 1429 of credential theft incidents in 2015 involved malware, particularly the ransomware malware (Reavis, 2012). Malware incidents have the following impact on organisations and computer users (Wlosinski, 2015), such as the following examples:

- Loss of data to governments, commercial businesses, financial institutions and individuals
- Theft of Identity (e.g. stolen credit cards, theft under the names of those affected)
- Online fraud (e.g. theft of account holdings by deception)
- Computer extortion (e.g. Ransomware)
- Unauthorised access to networks and personal computing and devices
- Distributed Denial of Service (DDoS) attacks against networks and websites of businesses and government, rendering their systems unavailable
- Data availability and business continuity that is affected by data destruction

Organisations are using ineffective techniques or systems to detect malware. Antivirus (AV) solution is one common way of detecting malware that uses its malware signatures database derived from analysing the malware code or instructions. However, any minor alterations on the malware code by malware writers can greatly prevent the malware from being detected. Because of this, malware and malware detection are in a mutual arms race, which is why it is important to concentrate rather on the detection and analysis techniques and environment (as shown below in the research question statement), rather than on the malware itself, as that changes almost daily (See 1.2. and 1.3 below).

## **1.2 Research Question**

How does one effectively perform Malware analysis and detection on enterprise systems in order to reduce the damage of malware attacks on the operation of organizations?

The following sub - questions are derived from the research question:

1. What are the malware analysis and detection techniques that are available in literature?
2. What are the tools and techniques available to effectively perform analysis and detection?
3. How to set up a malware analysis and detection environment?
4. How to measure the most effective technique of malware analysis and detection on enterprise systems?

The next section will discuss the objectives where the answer to the above questions will be explored.

## **1.3 Objectives of the Research**

The main research objective can be broken down into the following sub-objectives:

- To study and compare malware analysis and detection techniques that are in literature
- To investigate tools and techniques that are available to effectively perform malware analysis and detection
- To determine the most effective way of setting up a malware analysis and detection environment
- To measure the effective techniques of malware analysis

The findings of this research will hopefully contribute positively to organisations within the malware analysis fraternity.

## **1.4 Aim of Research**

The main aim of this research is to study the important aspects of malware analysis in literature and to determine the most effective techniques to perform malware analysis and detection.

## **1.5 Significance of Study**

The significance of the study is to reduce the spread of malware to a minimum in the enterprise systems for organisations.

## 1.6 Document and Chapter Structure

This dissertation consists of five chapters in addition to references and appendices. All the files and experiment results included in this document are made available electronically with this dissertation, and additional software, log information results of experiments are available on CD and USB sticks. This five chapter dissertation is structured as follow:

- Chapter 1 - Introduction

This chapter provides an introduction into malicious software and malware analysis.

- Chapter 2 - Literature review

This chapter describes the following:

- the background and theory to research of malware
- describes the different forms of malware
- the different purposes of malware
- malware analysis
- the different stages of analysis and the techniques used
- describes how to configure an isolated environment to be able to conduct malware analysis in a safe environment and the tools required for malware analysis

- Chapter 3 - Research Methodology

The Research Methodology chapter describes the methodology used in this dissertation.

- Chapter 4 - Malware Analysis Lab and Experiments

This chapter describes the practical experiments, results and investigations done by the author of this dissertation.

- Chapter 5 - Experiment Results Discussion and Conclusion

The discussion and review of the results presented from the experiments done in chapter 4 relative to existing research and knowledge.

## 1.7 Chapter Summary

The foundational background was established by discussing the introduction and background to this research study. The problem statement, objectives, aim of the research and significance of the study are also outlined in this chapter. The structure of chapters for the entire dissertation or research is also provided. Table 1 provides a summary overview of the research questions and objectives, as well as the chapters containing and addressing these questions and objectives.

<b>PRIMARY RESEARCH QUESTION</b>		
How is malware detection done through malware analysis on enterprise systems?		
<b>Secondary Research Questions</b>	<b>Secondary Research Objectives</b>	<b>Chapters</b>
What are the malware and detection techniques that are in literature?	To study and compare malware analysis and detection techniques that are in literature.	Addressed in Chapter 2, Literature Review
What are the tools and techniques available to effectively perform analysis and detection?	To investigate tools and techniques that are available to effectively perform malware analysis and detection.	Addressed in Chapter 2
How to setup a malware analysis and detection environment?	To determine the most effective way of setting up a malware analysis and detection environment.	Addressed in Chapter 2
How to measure the most effective technique of malware analysis and detection on enterprise systems?	To measure the effective techniques of malware analysis.	Addressed in Chapter 5

Table 1: Summary overview of research questions and objectives

The following chapter provides a detailed description of the literature review that relates to this research.

# CHAPTER 2

## LITERATURE REVIEW

### CHAPTER OVERVIEW

This chapter focuses on the following areas:

- Introduction
- Malware and History of Malware
- Malware Analysis
- Malware Detection Techniques
- Malware Anti-Reverse-Engineering
- Malware Analysis Lab
- Chapter Summary

The objectives of this chapter are as follows:

- To study and compare malware analysis and detection techniques that are in literature
- To investigate tools and techniques that are available to effectively perform malware analysis and detection
- To determine the most effective way of setting up a malware analysis and detection environment

### 2 INTRODUCTION

Anyone that uses a computer may have a direct or indirect encounter with malware, especially if the computer is connected to the Internet as mentioned in the previous chapter. In chapter 1 we discussed malware or malicious software as being a major problem to individuals and organisations and which also continues to grow.

This chapter focuses on the theory around malware and malware analysis in order to get a better understanding of malware as well as the process of analysing or dissecting the malware and how to setup a secure environment to analyse malware. This chapter also aims to provide answers to the following research questions:

- What are the malware and detection techniques that are in literature?
- What are the tools and techniques available to effectively perform analysis and detection?
- How to setup a malware analysis and detection environment?

## **2.1 Introduction to Malware**

As already described in Chapter 1, malware is malicious software of which the sole purpose is to attack and damage, disable or disrupt computers, computer systems, or networks, and can also steal information. Attackers or malware authors often take advantage of security flaws in systems or websites that are also called vulnerabilities, to infect by injecting malware into existing software and systems with the most severe consequences that can lead to incidents such as identity theft and/or financial gain. Computer users already know the damage that computer viruses can cause on infected computers but this does not mean that the malware and viruses are the same.

The term malware is an umbrella that encompasses a wide range of threats such as viruses, botnets, worms, spyware, trojans, backdoors and other malicious software. Each of the malware sub-types consists of unique characteristics, behavioural patterns, functionality and targeted systems. The sections that follow provide an in-depth study into malware and its different types.

## **2.2 Malware and History of Malware**

There are different types and categories of malware that are available. This section provides an overview of the history of malware, their characteristics, method of infection and the potential impact that these malware types might have on infected systems. It can be described as a program whose purpose is malevolent; to intentionally cause harm or subvert the intended function of the system, and it ultimately performs actions that were not sanctioned or agreed to by the user.

A noticeable incident was the spread of the Melissa virus that was designed to send emails to Microsoft outlook users. Every user or computer that was infected would, in turn, infect other users or computers (Milovsevi'c, 2013). The virus proliferated rapidly and exponentially, resulting in substantial interruption and impairment of public communications and services.

### **2.2.1 History of Malware**

During the early days of computers viruses and worms were created for fun and to perform maintenance functions on computers. Malicious software or viruses did not surface until the early 1940s when the first theoretical malware was discovered in 1949 by John von Neumann. The malware was believed to be self-reproducing automatons malware. Malware strains or variants have come a long way since 1949. The following provides a brief history of malware that has been discovered over the years:

- 1978, the first Trojan was released

- 1986, the first IBM compatible virus called “Brain”
- 1988, the first worm called “Morris” distributed over the internet
- 2003, the Slammer Worm infects 75000 machines within 10 minutes
- 2010, the Stuxnet attacks Iran Nuclear centrifuges

Malware code and function has evolved over the years as indicated in figure 2 below,

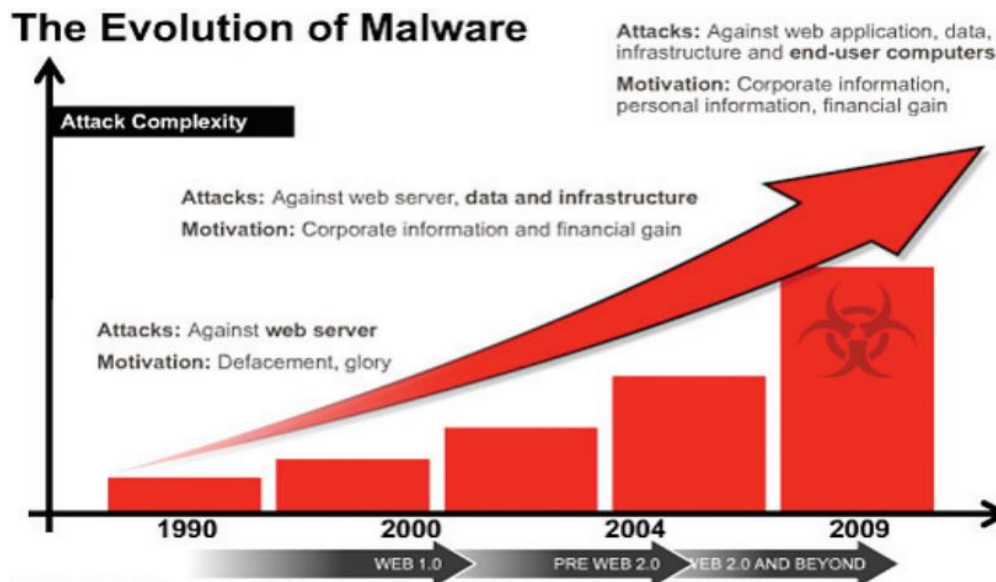


Figure 2: Evolution of Malware (Touchette, 2016)

## 2.2.2 Malware Types

The following are malware types as described by various sources (Elisan, 2015; Agrawal et al., 2014; Michael Davis, 2010; Privacy, 2008; Bhojani, 2014; Skoudis and Zeltser, 2003; Kendall, 2007; Sikorski, 2012; Kaspersky, 2016; Reavis, 2012; Marak, 2015):

- **Backdoor:** Its malicious code installs itself onto any system to allow the attacker access. Backdoor malware typically allows the attacker to connect the targeted system with little or no authentication and can execute commands locally on the system.
- **Botnet:** A botnet is different from a backdoor in that with a backdoor, the connection is from the outside to the inside, but with a botnet, the connection (after initial infection) is from the inside to the outside. All computers infected with the same botnet receive the same instructions from a single command-and-control (c&c) server.
- **Trojans:** Trojan horse is similar to the Greek myth that present themselves as harmless, useful software in order to persuade victims to install them on their computers; while Trojans naturally seem to be normal software, which is frequently bundled with other valid software that can introduce backdoors, allowing unauthorized access to your computer. Trojans do not attempt to insert themselves into other installed applications or system files like computer viruses do. Instead, they employ maneuvers such as drive-by

downloads or installing via online games in order to reach their targets (Agrawal et al., 2014). Once a trojan is activated on the victim's system, Trojans can enable cyber-criminals to spy on you, steal your sensitive data, and gain backdoor access to your system. These actions can include:

- Deletion of data
- Blocking of data
- Data modification
- Copying of data
- Disrupting the service and performance of computers or computer networks

According to Kaspersky (Kaspersky, 2016), trojans are classified according to the type of actions that they can perform on your computer:

- Backdoor
- Exploit
- Rootkit
- Trojan-Banker
- Trojan-DDoS
- Trojan-Downloader
- Trojan-Dropper
- Trojan-FakeAV
- Trojan-Game Thief
- Trojan-IM
- Trojan-Ransom
- Trojan-SMS
- Trojan-Spy
- Trojan-Mail finder
- Other types of Trojans include:
  - Trojan-ArcBomb
  - Trojan-Proxy
  - Trojan-Notifier
  - Trojan-PSW
  - Trojan-Clicker

- **Downloader:** Malicious software that is intended to only download other malicious software. Downloaders are normally installed by attackers when they have gained access to the targeted system. The downloader application downloads and installs additional malicious software to perform further malicious activities
- **Information-stealing malware:** It is malware that can collect information from the targeted computer and normally sends back to the controller or attacker, e.g. password



hash grabbers, sniffers and physical or software key loggers. This malware is also used for gaining access to online banking accounts such as email

- **Launcher:** Malicious applications that are used to launch other malicious applications. They usually use non-traditional methods to launch other malicious applications in order to prevent detection on the infected system
- **Rootkit:** Malicious code that is intended to conceal the existence of other malicious code. This type of malware is normally grouped with other malware, such as backdoors that allows an attacker to gain remote access and make the code undetectable on the infected system
- **Scareware:** It is malware that is intended to scare the infected user into purchasing something. It commonly has an interface that makes it look like an antivirus solution. It tells the users that there is malicious software installed on their system and the only way to remove the software is to buy their malicious software when in fact, the software that is being sold has nothing to do with the removal of the scareware
- **Crime ware:** The rise of ransomware and Ransomware-as-a-Service (RaaS) has been the biggest change to the landscape of malware in recent years. Ransomware makes use of infection methods such as
  - spam
  - social engineering methods
  - drive-by downloads
  - malvertisingIt encrypts all the data on a targeted computer and holds the data for ransom, typically to be paid using bitcoin (Agrawal et al., 2014).
- **Spam-sending malware:** This type of malware infects a computer and then uses that computer to send spam emails. This malware generates income for attackers that allow them to sell email spam sending services.
- **Worm or virus:** Virus consists of harmful applications intended to infect legitimate software applications. Once a person installs and runs the infected application, the virus triggers and disperses itself to other applications that are installed on the computer before performing further malicious activities such as deletion of critical system files within the operating system. Worms are stand-alone applications that are able to replicate themselves across the network. Unlike a virus, a worm does not need to attach itself to an existing application. On the other hand, both viruses and worms can severely damage the infected computer as they are able to exploit network shares (Agrawal et al., 2014).
- **Shadyware, PUPs, Adware and Keyloggers:** Shadyware is the type of malware that does not technically fit into the category of viruses because it is identified as “potentially

undesirable processes” (PUPs). It may still invade user’s privacy, contain malicious code, and even become a nuisance (Agrawal et al., 2014) on the infected system.

Adware is a type of financially motivated malware that normally appears in the form of unwanted advertisements presented to the user. Websites are filled with these types of applications that can hijack a targeted computer for profit. Most of the adware is normally bundled with other software known as “free” downloads and pop-up toolbars or ads that unnoticeably install software on targeted computer with vulnerabilities (Agrawal et al., 2014).

Spyware/keyloggers constitute another type of malware that secretly collects information on the infected computer and sends it back to the attackers’ computer. Information collected includes websites that the user visited, system and browser information. Spyware is usually installed by a trojan and does not also have any infection traits. Once installed on the targeted computer, it then starts collecting information quietly in order to avoid detection. Keyloggers are variants that record the key strokes entered by a user, and it is common in point-of-sale (POS) and web application attacks (Agrawal et al., 2014).

- **RAM Scrapers:** RAM scraping is the type of malware that is intended to steal payment information from compromised POS devices. This malware’s threat vector exploits vulnerabilities in the transaction process where information is stored in unencrypted form in the system memory for just a few milliseconds. RAM scrapers use this period to snatch card information during each transaction and saves it as a text file for exfiltration at a later date (Agrawal et al., 2014).

### 2.2.2.1 Summary of Malware Types

Table 1, below is a summary of the malware type section, the purpose of this table is to provide an overview of the section on the types of malware described in section 2.2.2 above.

Types of malware	Feature	Mode of operation	Damage caused
Virus	A form of malware that takes unauthorized control of the infected computer and causes harm without the knowledge of the user.	Viruses attach themselves to a program such as an executable file and its self-replicating capability spreads the infection from one computer to another.	Causes denial of service performance degradation.

Types of malware	Feature	Mode of operation	Damage caused
Worms	Worms are standalone malicious software that can operate independently and does not hook itself to propagate.	They exploit the security vulnerability by using computer or network resources and spread themselves via storage devices such as USB devices, communication media such as Email.	Cause network performance issues, consumes large amount of memory of systems resources.
Trojan	Malicious piece of software that conceals itself and behaves as a legitimate program to take unauthorized control of the computer.	Trojan does not self-replicate; instead is downloaded through user interaction such as downloading a file from the internet.	Steal password or login details, electronic money theft, modify/delete files, monitor user activity.
Rootkits	Rootkits are the masking techniques for malware, basically designed to conceal the malicious software.	Can be installed through a software exploit or by a Trojan.	Steal passwords, install Keyloggers.
Spyware	A software negatively affecting a system by keeping track of users' activity without their consent and send back the sensitive information to the creator.	Can be installed with other software such as freeware or dropped by Trojans.	Some sophisticated type of spyware captures entire network interface, digital certificate, encryption keys and other sensitive information.
Keyloggers	Serious form of Spyware secretly record keystrokes, read cookies and files on the drive to gather personal details.	Can be installed by another malicious program or when a user visited an infected site.	Capture sensitive information such as username, password, credit card number or online banking details

Table 2: Malware Types Summary (Uppal et al., 2004)

### 2.2.3 Overall Characteristics of Malware

Malware provides attackers with a convenient, ease of use and automation means of compromising computer systems. Malware normally consists of the following characteristics (Privacy, 2008):

- Multi-functional and Modular
- Persistent and Efficient

- Can Affect a range of devices
- Part of a broader cyber-attack system
- Profitable

#### **2.2.4 Categories and Behaviour of Malware**

The majority of malware have similar characteristics, behaviour and properties that can be detected by a majority of antivirus solutions using signature detection methods. A worm is a good example to use as a worm self-replicates and spreads itself by making copies of itself on the infected system through the available communication networks. A virus will try to spread by making use of a carrier such as attaching itself to a file.

##### Environments of malware.

For malware to perform malicious activity on an infected system and also the neighbouring system, some resources should exist for the malware to succeed. Malware authors develop their code to execute on a specific operating system (OS). For example, a virus written to execute on a Microsoft OS may not execute on a Linux or Unix OS. The malware may require some specific applications to be running on the infected system in order to be effective. Malware can be written to execute when it detects the Microsoft office application on the system and then make use of scripting language and macros.

Ways of infection. Malware uses various ways of infection on a system such as flash drives (USB), which could be infected with malware, and these flash drives are plugged in on a different system and also can be transmitted on email attachments.

Each type of malware has its own unique behaviour towards the infected system. This behaviour is developed as program code and embedded or obfuscated within the payload of the malware. By examining the malware payload we can determine its behaviour, which can either be malicious or non-malicious, and the threat it may pose on the system it has infected.

#### **2.2.5 Malware Functionality**

This section discusses the following:

- Malware Behaviour
- Deployment of malware
- Malware Lifecycle Model

### 2.2.5.1 Malware Behaviour

Malware behaves in various ways when executed on a target system and the behaviour is discussed in detail in this section.

- **Downloaders and Launchers**

The two types of malware that are normally stumbled upon are downloaders and launchers. *Malware downloaders*: downloads another piece of the malware from source and executes on the target systems. These types of malware are often packed with exploits that ordinarily use OS API's such as URLDownloadToFileA followed by a call to WinExec to download and to also execute the downloaded malware (Kendall, 2007).

*Malware Launchers*: are also known as loaders which are any executables that install malware for instant or impending concealed execution of malware (Kendall, 2007)

- **Backdoors**

Backdoors are types of malware that allow an attacker with access remotely to a compromised system and they are the most common type of malware that have a variety of abilities. Backdoors initiate network communication to the internet in a number of ways and the mostly use HTTP protocol on port 80. The HTTP protocol is mostly used for outbound traffic, which is the traffic that leaves the organization to the internet, and allows malware to take advantage of this process (Kendall, 2007). Backdoor malware consists of certain functions that enable the malware to

- ✓ modify the registry
- ✓ enumerate display windows
- ✓ create additional directories
- ✓ search for files

- **Credential Stealers**

Malware authors go to excessive lengths to write malware that can steal user's credentials and there are three types of malware that can perform this function (Kendall, 2007):

- a) Malware that initiates when a user logs on the target system.
- b) Malware that can make a copy of the information stored on the target system such as hashes of passwords that can be used to perform further attacks.
- c) Malware that records every keystroke executed.

- **Persistence Mechanisms**

As soon as malware has infected the targeted system, its intention is to root itself on the system and stay undetected for an extended period of time. The persistence process can serve as a way to find patterns of the malware. A common method of persistence is the modification of the registry keys on the system and modification of system files.

- **Privilege Escalation**

A common practice by users is to logon and run with local administrator privileges on systems that allow malware to be executed successfully. Some malware can use flaws in the OS or application to gain root/admin privileges even though the user whose credentials are being used does not have those privileges. What is described here is credential stealing (if the user is already logged on with privileged account) rather than privileged escalation. Administrator privileges also allow malware executed to have the same administrative privileges. Security best practice of setting up and login on any system is to not run as or be part of the local administrator group. If malware is executed on the system without administrator privileges, then the malware would need to have the ability to perform privilege escalation on the target system. A bulk of malware that perform this type of attack are commonly known as exploits or zero-day malware attacks against the system (Kendall, 2007).

- **Covering its Tracks**

Most malware authors ensure that the malware that they design is undetectable and can hide its execution through various mechanisms already discussed. The malware used often that hides its malicious intention, is called a rootkit. This type of malware modifies the internal functioning of the system and also embeds itself in the systems (Kendall, 2007).

### **2.2.5.2 Deployment of malware**

Malware is deployed to the target system through various means of which some are authentic ways while others are malicious. Deployment of malware is as important as the capabilities or functions of malware and this process does not concern itself with how the malware is executed on the target system but how it reaches its target.

Some deployment methods have the ability to execute the malware that can result in deployment and infect the targeted system. The methods or technologies used for the deployment process of malware are known as malware infection vectors (Elisan, 2015).

#### **2.2.5.2.1 Malware Infection Vectors**

This section discusses what malware infection vectors are and how they are used and abused by malware authors to deploy malware on their target systems. Malware infection vectors are liable for the delivery and propagation of malware in a threat ecosystem (Elisan, 2015).

Threat ecosystem is defined as a collection of different technologies that attackers use to perform attack operations (Elisan, 2015). Malware infection vectors are selected based on various criteria that include the following:

- a) Speed
  - b) Covertness
  - c) Exposure
  - d) Period of Time
- **Speed:** the speed in which the malware infects a target system is important for malware authors and attackers. The speed of infection depends on the intention of malware or author; if time is critical then the faster method is chosen. Historically malware infection depended mostly on physical media as infection types, and nowadays it depends on inter connection and communication of systems (Elisan, 2015)
  - **Covertness:** the success of malware infection vectors is the one that is undetectable and bypasses most security controls in place. Software vulnerabilities are the stealthiest of types of malware infection vectors. Well written malware can exploit a vulnerable software by taking advantage of its privileges and access on the target system. Speed and Covertness (stealth) are opposed to each other. The more speed a virus has (the quicker it propagates), the more noise it makes and the less covert/stealthy it is.
  - **Exposure:** The huge number of targets that a malware infection vector can infect in any given time (Elisan, 2015) is critical in performing successful attacks. Phishing and spam emails that are carrying malware or even malware that is downloaded are also used as an infection vector.
  - **Period of Time:** Some malware are designed to expire or terminate itself after a certain period and a typical example will be vulnerable software that when patched, will render the exploits useless; this will make the deployment of malware unsuccessful.

#### 2.2.5.2.2 Types of Malware infection Vectors

Malware is deployed using one or a combination of the following infection vectors:

- a) Physical Media
- b) Email or Attachments
- c) Chats and instant messaging
- d) Social networking
- e) URL links
- f) Websites
- g) File share
- h) Vulnerabilities found on software
- i) Vulnerabilities found on common protocols

## 2.2.6 Malware Lifecycle Model

Malware frequently follows a methodical lifecycle that assists to understand its resilience. Figure 3 below, proposes a malware lifecycle model, covering five areas or phases that malware will typically undergo.

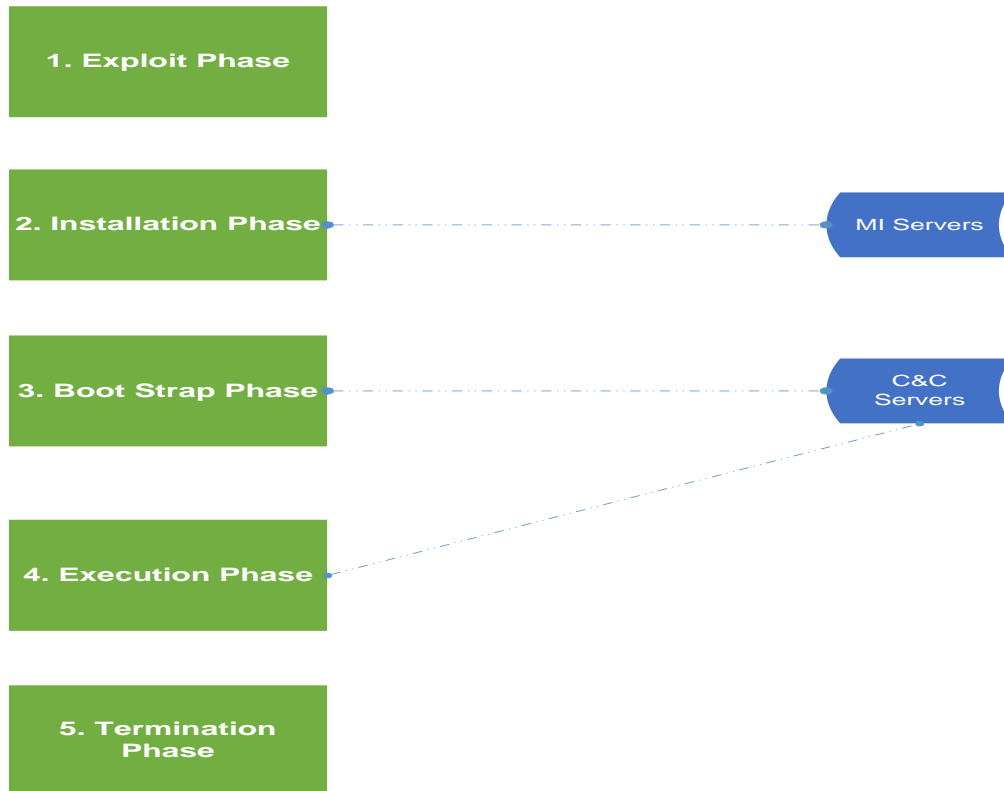


Figure 3: Malware Lifecycle (Rossow, 2013)

The malware cycle proposed phases are described below:

### a) Exploit Phase

In this phase, a system that is vulnerable could be exploited by an attacker by gaining access to the targeted system. The techniques that are used to exploit the targeted systems are mostly client based vulnerabilities that include but are not limited to drive-by downloads. This is where vulnerabilities on the browser are exploited by malicious code. The exploits could normally transmit shellcode as payloads which are small pieces of malicious software that are executed immediately after the exploit. Otherwise, malware could spread in the following ways:

- Through spam attachments
- Infected Files
- Memory sticks
- Network traffic



**b) Installation Phase**

In this phase, the malware that carries the full functionality is often downloaded from the installation of MI server and it is then installed on the targeted system. Thus, server renders malicious code or binaries which is a clear indication that they play a critical role in the infrastructure of malware

**c) Boot Strap Phase**

In this phase, we see that once the binaries have launched, the malware will then begin to initialize by making contact with the C&C environment. If the malware achieves to connect to the C&C environment then the malware becomes part of bots or C&C network. This will allow the malware to be remotely controlled from the C&C main system

**d) Execution Phase**

In this phase, the C&C environment will send communication commands to the bot or infected system. The main C&C server will then perform further malicious activities on the infected system such as spamming or stealing data

**e) Termination Phase**

If the malware is then detected or if the communication to the C&C is terminated, then the malware will terminate itself in this phase

## **2.3 Related Work**

Previous research has indicated that malware analysis is vast and well documented (Agrawal et al., 2014; Bhojani, 2014; Saffaf, 2009; Valli and Brand, 2008; Distler and Hornat, 2007; Sikorski, 2012; Elisan, 2015). The malware analysis is the process that seeks to uncover the functionality, purpose and to determine how malware had infected a targeted system. Many malware is normally detected and stopped by antivirus applications and other similar solutions or tools that mostly fail to detect new variant or obfuscated malware. The most visible reason is that majority of these solutions use signature based detection capabilities. These solutions compare the code of the suspected malware sample against its signature database where each signature code is able to identify the malicious pattern of the code. However, obfuscation methods such as polymorphism and metamorphism hinder the signature based solutions by randomly encrypting the malicious software code in a way that does not compromise the original functionality of the software(Bhojani, 2014; Saffaf, 2009).

Once any trace of malicious code is detected, malware analysts and researchers will start to analyze and dissect the malicious code. The malware analysis process is achieved by reversing or dissecting the malware or malicious code using the two malware analysis techniques and tools that will be discussed in detail later in this Chapter. Static analysis is the process of analyzing the malware code without executing it while Dynamic analysis is the

process of executing the malware in an isolated virtual sandbox environment and its behaviour is monitored and recorded (Elisan, 2015; Valli and Brand, 2008; Distler and Hornat, 2007).

The virtual sandbox environment does not contain any critical data and must not be connected to the production network. The network traffic from the virtual environment must be redirected to the host system or a virtual network must be created in order to contain the network traffic generated by the malware. There are several virtual solutions that exist such as VMWare, Virtual-pc and Hyper-V (VMWare, 2016; Virtualbox, 2016; Microsoft, 2016). These solutions assist malware analysts or researchers to be able to analyze and dissect the malware in a secure environment. These virtualization solutions allow guest systems with operating systems to be installed on a physical host and these guest systems are known as virtual machines. The virtual machines are created by the virtual software to intercept the access to the physical host's hardware and features. These systems or machines running on the physical host behave the same way as a physical system and it consists of a virtual central processing unit (CPU), random access memory (RAM), hard disk and also a network adapter (VMWare, 2016; Virtualbox, 2016).

## **2.4 Malware analysis**

Malware analysis is defined as the process of dissecting malware to understand how it works using static and dynamic inspection of various tools, methods and processes in order to understand how to identify malware, and how to defeat or eliminate malware (Sikorski, 2012). This process is an essential step in developing effective detection techniques for fighting malicious code and it is also an important requirement for developing tools that can eradicate malware from an infected system (Savan Gadhiya and Bhavsar, 2013).

Information is gathered by dissecting malware using extraction and monitoring tools; the methods and processes that are required to successfully collect information about the malware differ depending on the behaviour and capability of the malware. Various tools, methods and processes are used to extract information from the analysed malware without disassembling it, which allows malware to be analysed in an isolated controlled environment for the purpose of collecting and monitoring information that can be used to reveal the malware's true intentions.

Disassembling is defined as the process of breaking down binary into low-level code such as assembly code (Elisan, 2015). The process of malware analysis consists of two types, static and dynamic that are explained in detail in the next section.

### 2.4.1 The goals of Malware analysis

The goal of malware analysis is to (Valli and Brand, 2008):

- gain insight into nature and purpose of Malware
- identify host-based and network indicators of compromise (IOC)
- understand malware behaviours and its persistence mechanism
- extract information used for learning and malware detection
- try to determine the origin of the authors of the malware (in cases of nation-state malware)

Information collected from performing malware analysis on a malware sample or infected system also provides an understanding of the malware's behaviour and capabilities and how to detect it on the targeted network or system. The Malware analysis process should produce an analysis report that will allow us to perform the following actions (Elisan, 2015):

- Prevent the spread of malware - Preventing malware from causing further damage is usually by determining how the malware infected the system initially. Understanding the vectors of infection that are used by malware allows the preventing and remediating of the method that was used to compromise the infected system. It is often difficult to determine the infection vector of malware and in such cases, it is best to understand how the malware behaves on an infected system and prevent that from further compromising the network
- Detect the Presence of Malware – Information that is collected from performing malware analysis is used as methods to detect the malware on the enterprise network. The most common methods of detecting malware on an infected system are as follows:
  - System changes
  - Extract of the code that is collected from the analysed malicious code
  - This would normally reside in memory
  - Certain strings/behaviour/pattern picked up in network traffic

System changes which are applied by the malware, are used to detect the presence of the malware artifact. Code extracted from the malicious code can be used to create a signature of the malware that is a practice often performed by Antivirus vendors. The code used to create malware signatures must be from the extracted code that is not encrypted and thus will create false alarms.

False alarms consist of two types that are:

- a) False Positives - It is when a legitimate file is inadvertently being detected as infected, malicious and/or suspicious (Mishra, 2013)

b) False Negatives - It is when a file that is malicious is miscategorised as a clean file. This is maybe due to similarities between applications or files that are well known as malware (Mishra, 2013)

- Remediate Malware Infections - When the malware has been detected on an infected system or file the next critical step is a remediation process. This is where the malware analysis process becomes important, as the malware analyst will be able to identify IOC and be able to perform remediation on the infected system

In some instances it is difficult to perform remediation as the malware could have rooted itself deeply in the system and any steps of performing remediation may then damage or corrupt the system even further, which can reduce the system to a state that can make it inoperable. This could even lead to rebuilding the system entirely. There are tools that can perform remediation on the infected system and these tools can either synthesise system changes common to malware or synthesis changes done by specific malware strains.

#### **2.4.2 Challenges of Malware Analysis**

Malware analysis is a challenging task when the source code of the malware is made available and it is even more challenging when the source code or debug information is not present. We have mentioned in the previous sections that malware authors often create malware with countless evasion techniques to obstruct or block the static and automated analysis process. This makes it challenging to reveal the malware's intent and the full scale of its hidden capabilities (Saidi, 2012).

#### **2.4.3 Limitations of Malware Analysis**

Despite the malware analysis process being a critical process in understanding the true intention of malware, it also has its own limitations. From a static analysis point of view the information extracted from malware is effective only when the malware's actual form is not obfuscated, hence why DE obfuscation and unpacking of malware is key (Elisan, 2015), as they remove the hindrance of statically analyzing malware.

This limitation is eased when malware which is obfuscated, is analyzed by various tools and methods that assist in revealing the obfuscated code of malware. While dynamic analysis is all about executing malware in an isolated environment, its limitations are because of the dependencies of malware that enable it to execute on a targeted system.

The dependencies of malware executing include the following:

- Applications

- Users
- Environment
- Timing
- Event or logging

If none of the above mentioned dependencies are available or satisfied, then the malware will not execute successfully or may partially execute on the targeted system. You often find that malware does not execute if some or all dependencies are not satisfied (Elisan, 2015) and this will make reporting on dynamic analysis inefficient.

#### 2.4.4 Malware Analysis Types

Malware analysis is defined as the process of dissecting malware to understand how it works. Malware analysis has been traditionally known as a manual process that is tedious and time consuming and the number of samples that need to be analysed on a daily basis by security vendors is constantly increasing (Savan Gadhiya and Bhavsar, 2013). This process uses static and dynamic inspection of various tools, methods and processes in order to understand how to identify malware, and how to defeat or eliminate malware (Sikorski, 2012).

The malware analysis process consists of two types of analysis which are static and dynamic analysis, and the reason for that is that malware can be static or dynamic. The process of analysing a program by inspecting its properties or code is known as Static analysis. The process of executing a program in order to analyse it is known as Dynamic Analysis.

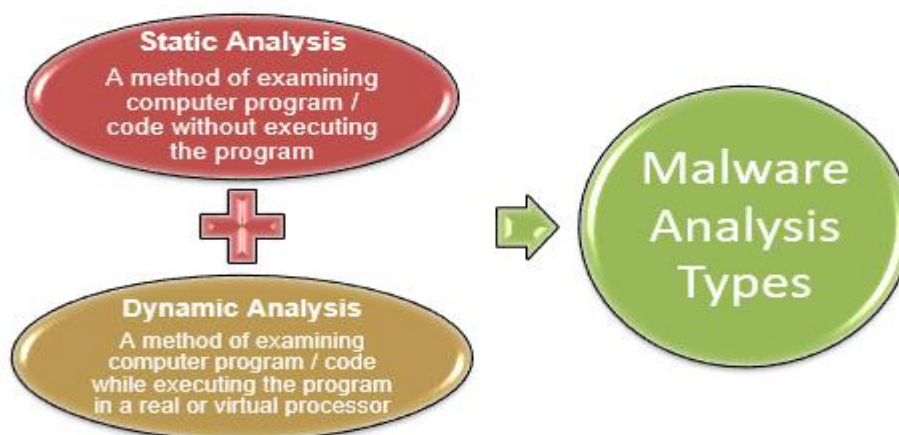


Figure 4: Summary Malware Analysis Types

The two types of analysis as indicated on figure 4 above are explained in detail in the following section.

#### 2.4.4.1 Static Analysis

Static analysis of malware is defined as the process of extracting information from malware while it is not running, by analysing the code of the malware to determine its true intention (Elisan, 2015). Extraction of information from malware includes the examination of disassembly listings, extracted strings, obtaining signatures of a virus, determining the architecture of the target and compiler that is used, as well as many other characteristics of malware. The malware program or artifact is not executed during this process, which then requires specialised tools in order to analyse the malware in detail.

Tools that are used to extract information from malware can assist to gather information such as file type, malicious non-encrypted strings or code that is normally found in the structure of a malware. Static analysis is regarded as the easier, quickest and less risky of the malware analysis processes; as previously mentioned malware is not executed while it is statically to be analysed. Another reason that static analysis poses less risk when analysing malware, is because of the tools that are available in other OS's running on Linux based platforms such as REMnux running on Ubuntu OS. The REMnux system consists of various tools that can be used to perform static analysis on malware by securely extracting malware information from malware destined to affect windows OS's or files.

According to (Savan Gadhiya and Bhavsar, 2013) there are several techniques used in the static analysis process which include, but are not limited to the following:

- File Fingerprinting: This includes computation of cryptographic hash of the binary in order to uniquely distinguish it from other binaries and to verify that it has not been tampered with
- Extraction of hard coded strings: There is software that typically prints output, for example status or error messages about the software hidden in the compiled binary as readable text. These embedded strings often allow us to be able to draw conclusions about the structure and the contents of the inspected binary.
- File format: Useful information can be gathered by leveraging metadata of a given file format. This includes the magic number on UNIX systems to determine the file type as well as dissecting information from the actual file format. For example, the windows binary which is naturally in a portable executable (PE) format, and a lot of evidence or information can be extracted from it such as the time it was compiled, imported and exported functions, strings, menus and icons
- AV scanning: If the malware sample is known, it is likely to be detected by one or more AV solutions

- Packer detection: Today's malware is often distributed in an obfuscated form for example in an encrypted or compressed format. This is accomplished using a packer application, while arbitrary algorithms can be used for alterations. After packing the application looks a lot different from a static analysis standpoint and its logic, as well as other metadata that is harder to uncover. Although there are certain unpackers, such as UPX or PEiD, there is accordingly no generic unpacker, making this a major challenge of static malware analysis
- Disassembly: The disassembly of a given binary is the key part of static analysis. This is steered by utilising tools that are capable of reversing the machine code to assembly languages, such as CFF Explore or IDA Pro applications.

#### 2.4.4.1.1 Limitations of Static Analysis

Usually, the source code of malware samples is not freely available. This reduces the static analysis process to those that gather information from the binary representation of the malware sample. Analysing binaries brings along complex challenges. The static analysis process is considered to be of low risk and less rewarding because it provides less information from what can only be observed from the structure or code of the malware while it is static (Elisan, 2015). This information also does not reveal the true intention, characteristics and function of malware as opposed to when it is executed. The static analysis process complements the dynamic analysis process that is explained in detail in the next section.

#### 2.4.4.1.2 Overview of Static Analysis Process

Figure 5, below provides an overview and a graphical view of the Static analysis process discussed in the chapter.

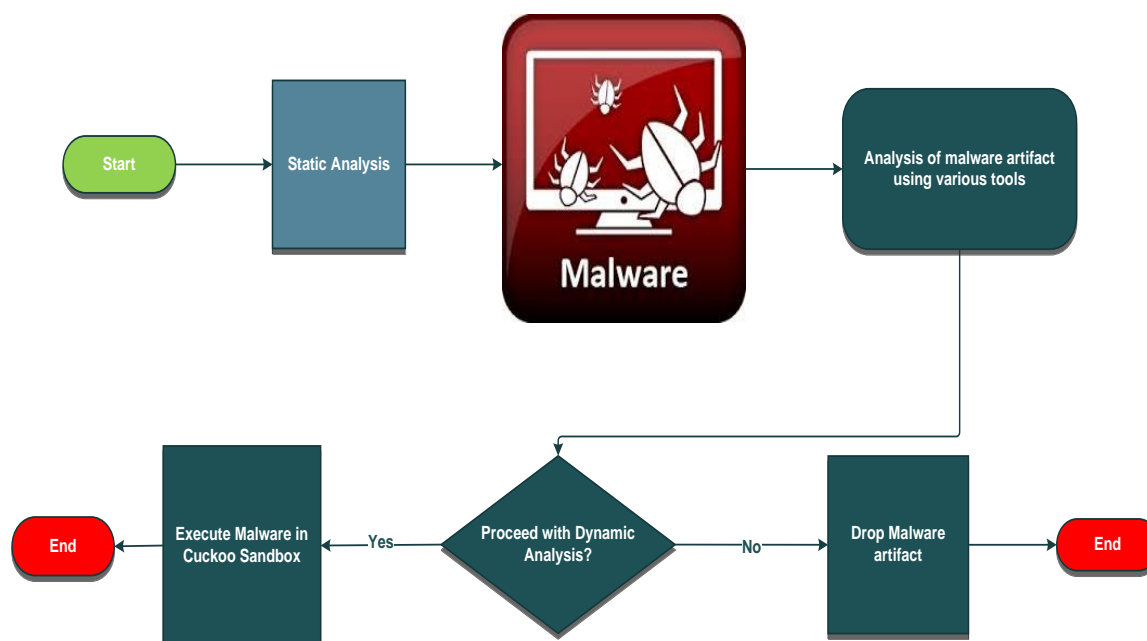


Figure 5: Overview Static Analysis Process

#### **2.4.4.2 Dynamic Analysis**

Dynamic analysis is defined as the process of extracting information from malware when it is executed (Elisan, 2015). This process entails executing the malware artifact in a secure isolated environment, unlike the static analysis process that provides only a view of the malware that is being analyzed. Dynamic analysis provides an exhaustive view on the malware's true intention, characteristics and function as information about the malware are gathered while the malware is running. Since Dynamic malware Analysis is performed during runtime and malware unpacks itself, this process evades the restrictions of static analysis, that is unpacking and obfuscation problems. It is thereby easy to see the actual behaviour of a program (Savan Gadhiya and Bhavsar, 2013).

Dynamic analysis uses automated tools such as the Cuckoo sandbox to analyze and monitor the malware that is executed in an isolated environment and allows analyses of malware at a large scale. Dynamic analysis is also known as behavior analysis process that follows an automated approach to analyzing malware (Zeltser, 2016) by using sandbox technology such as the Cuckoo malware analysis sandbox.

The malware isolated environment sandbox is a system that allows malware to be executed securely for analyzing malware without infecting other systems in the network. It is created to ensure that all the conditions for malware to execute are satisfied within the sandbox. The sandbox consists of an OS where the malware artifact would execute and the sandbox will also include all the dependencies required by the malware (Claudio Guarnieri, 2013).

The tools included in the sandbox allows the guest system to safely execute malware and monitors the system for any changes made by the malware. This can include network communications on various protocols, file and registry access and/or modifications, interaction with services or programs and other behavioural activities.

These system changes are recorded by the sandbox and a report will then be made available on the behaviour or interactions that the malware made with the infected system. The sandbox system gives consideration to the services to emulate for the network based malware to interact with, for the behaviour of the malware to be observed. The malware that infects a system may simply be the first stages in the process of infection as the malware maybe perform attempts to download the payload as the second stage of infection.

Dynamic analysis is considered to be a very risky and highly rewarding process as the risk of infection of a malware that is executed is very high as the infection can spread to other



production or critical systems in the enterprise network. The rewarding process of dynamic analysis allows the malware to reveal its true intention or function of the malware that is executed.

According to (Savan Gadhiya and Bhavsar, 2013) there are two key basic approaches for distinguishing dynamic malware analysis viz.:

- Analyzing the difference between defined points: When the acquired malware sample is executed for a specific period and afterwards the changes made to the system are then analyzed by comparison to the initial system state. The report generated will then state the behaviour of the malware that is being analyzed
- Observing runtime-behavior: Malicious activities launched by malicious applications are monitored and recorded during runtime using specialized or automated tools.

#### 2.4.4.2.1 Limitations of Dynamic Analysis

Dynamic analysis provides only a partial “effects-oriented” side view of the full potential of a given malware binary. Dynamic analysis does not reveal the effects of programming logic that fail to execute during the runtime process or analysis (Idika and Mathur, 2007).

#### 2.4.4.2.2 Overview of Dynamic Analysis Process

Figure 6, below is intended to provide an overview and summary of Dynamic Analysis area.

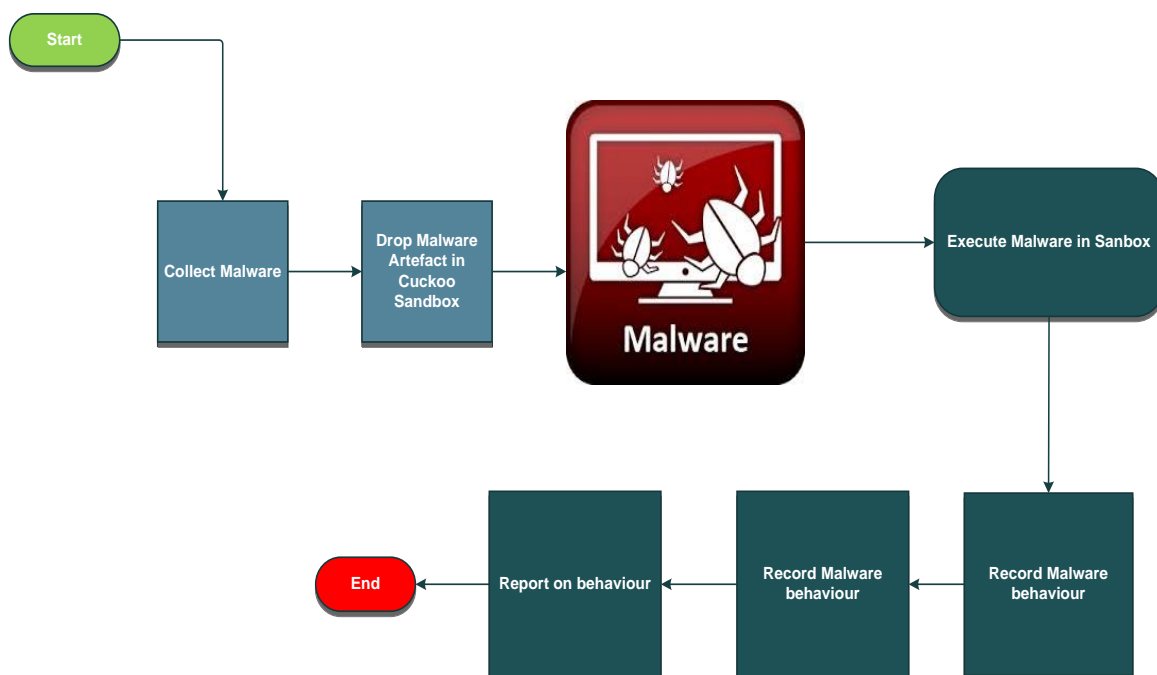


Figure 6: Overview Dynamic Analysis Process

## 2.4.5 Malware Analysis Process Overview

Figure 7, below is intended to provide an overview on the entire malware analysis process summarizing the sections 2.4.1 to 2.4.5.

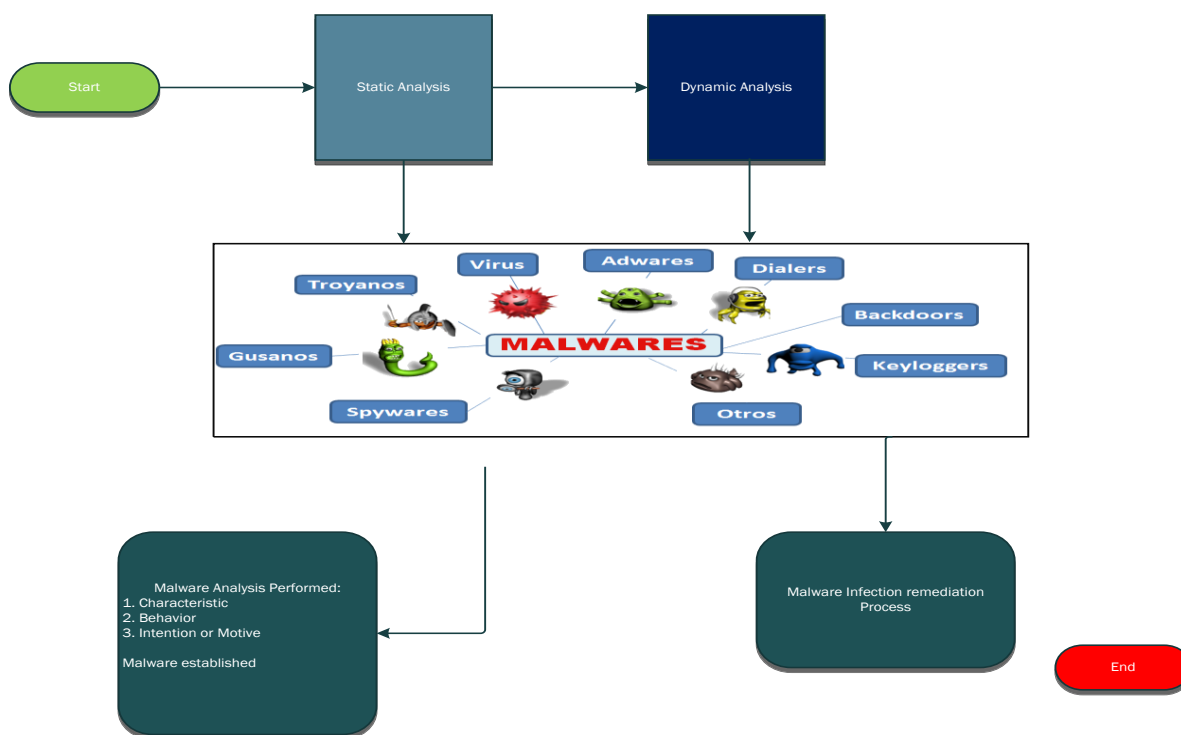


Figure 7: Malware Analysis Process overview

## 2.4.6 Protective Mechanisms

When malware is executed it has full access to its source code which means that when the malware is executed it will make use of the protective features that are built within its code in order to protect itself from detection (Elisan, 2015). The following are the common protective mechanisms available to malware when executed:

- Anti-sandboxing
- Anti-debugging
- Anti-virus
- Host-detection
- Network Behaviour

## 2.4.7 Malware Dependencies

The purpose of any malware is its ability to execute on any targeted system to achieve its objective. Malware is a software program like any other software that consists of dependencies

that it runs on. The more the malware is packed the more it will have a lot more dependencies it will rely upon to execute effectively.

These dependencies are also depended on the targeted system's characteristics and these dependencies can include but not limited to:

- **Environment:** The environment, can consist of Operating system, system settings and virtualization
- **Program:** Malware can make use of specific programs already installed on the targeted system
- **Event and / or Timing:** Some malware are dependent on a specific event or time to take place before they can execute on a targeted system
- **User:** Malware can also execute based on the user's access on the targeted system
- **File:** Malware that is designed to steal or use specific files installed on the targeted system

## 2.5 Malware Detection Techniques

Detection of malware is an area of great concern not only to the research or security community but also to the public. Techniques that are developed by security researchers or vendors that allow for detection of malware through the implementation and deployment of malware detectors. These malware detectors try to identify malware by detecting malicious behaviour. These techniques are categorized into two categories namely, signature and anomaly detection (Idika and Mathur, 2007).

### 2.5.1 Signature Based Detection

It is sometimes spoken of as a misuse detection that uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection (Shaban, 2013). Figure 8, below indicates that each technique employs one of three approaches that include Static, Dynamic and Hybrid.

Static analysis tries to detect the malware and its properties before the software executes, while Dynamic analysis tries to detect and monitor malicious behaviour during or after the execution of software. Hybrid analysis is a combination of both Static and Dynamic Analysis (Idika and Mathur, 2007).

Both dynamic and static analysis techniques have unique advantages and disadvantages. Dynamic analysis provides only a partial "effects-oriented" view of the full potential of a given malware binary. Dynamic analysis does not reveal the effects of programming logic that fails to execute during the runtime process or analysis.

Static program analysis provides a more comprehensive valuation of the entire code and data of the malicious software. For example, by analyzing the sequence of invoked system calls

and APIs, tracking data segment references and performing flow control analysis, it is possible to deduce temporal triggers, logical code bombs, and other malicious activities on the system and from there form a higher level semantics about the malicious behaviour. Features such as the presence of the object creation, the registry, OS manipulations, logic of network communication, and whether these capabilities are exercised during runtime or not (Shaban, 2013, Idika and Mathur, 2007).

## 2.5.2 Anomaly Based Detection

Typically occurs in two phases:

- Training phase in which the detector attempts to learn the normal behaviour
- Detection phase

The advantage of an anomaly based detection is its ability to detect zero-day attacks. The success of these methods relies on what functions should be learnt in training phase to distinguish malware and to also benign accurately (R and Rai, 2012, Shaban, 2013, Idika and Mathur, 2007).

### 2.5.2.1 Specification-based detection

This is a type of anomaly based detection that leverages some specification or rule, set of what is valid behaviour, in order to decide the maliciousness of the software that is being analyzed or inspected.

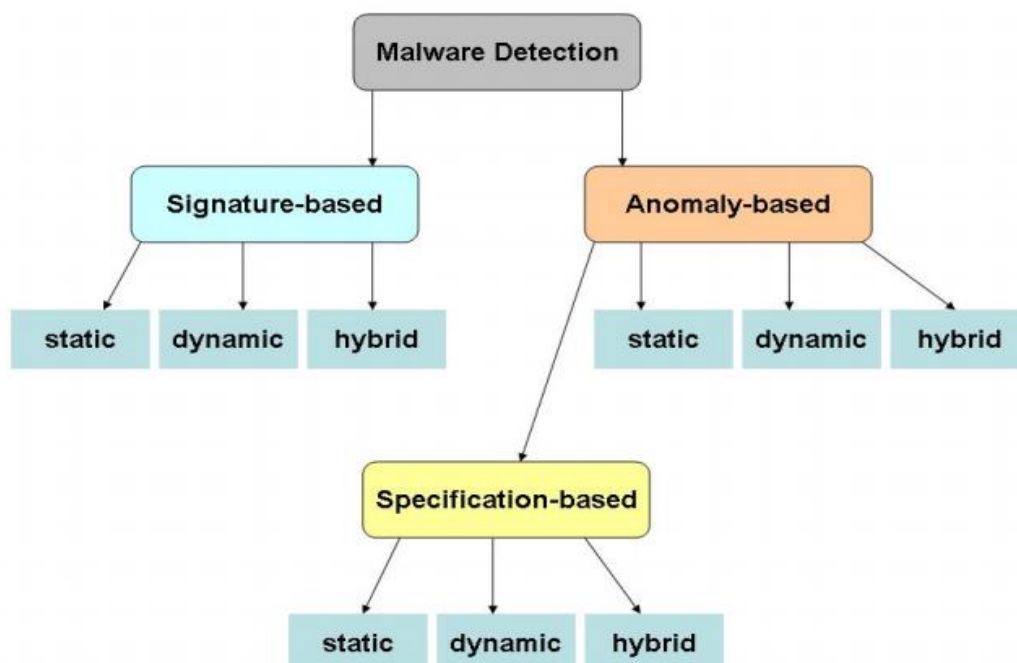


Figure 8: A classification of malware detection techniques (Idika and Mathur, 2007).

## **2.6 Malware Anti-reverse-engineering and anti-analysis**

### **2.6.1 Malware anti-analysis**

#### **2.6.1.1 Anti-virtual Machine**

Analysis of malware samples is recommended by Lenny Zeltser (Zeltser, 2010) to be performed on isolated environments (sandbox) such as virtual machines (VM). This will enable multiple VMs to be hosted on a physical host machine and the network activity from these VMs will also be isolated within the virtual network. Backups or snapshots of these VMs can be taken at any point in time and also restored rapidly. This makes it an ideal secure environment for performing analysis of malware where a known state of the VM can be restored at any time and the analysis process restarted whenever required. Analyzing malware in an isolated environment enables the close study of the behaviour of malware (Chen et al., 2016).

However, malware authors are now designing malware that is able to detect the environment in which it is running. Once the malware is able to detect the environment, the evasion mechanism within the malware may prevent the malicious software from running or even alter the malware's behaviour to avoid exposing malicious activity while running in a VM environment (Keragala and Walker, 2016).

For example: When the botnet malware is running on a physical machine it will attempt to connect to its Command and Control (C&C) server, but when it detects that it is running on a VM it will connect to a legitimate domain, causing the malware analyst or the security system to believe this is legitimate software. Malware authors use various methods or techniques to determine the environment that the malware is running on.

Some of the methods include (Keragala and Walker, 2016):

- Checking CPU Instructions
- Checking for Known Mac Addresses
- Checking for Registry Keys
- Checking for Processes Indicating a VM
- Checking for Existence of Files Indicating a VM
- Checking for Running Services

By reducing the presence of the above mentioned methods in the VM environments, it will make it harder for attackers or malware to identify and potentially bypass the VM or fool the malware analyst.

## **2.6.2 Malware anti-reverse-engineering**

### **2.6.2.1 Portable Executable**

There is a requirement to have an understanding of the executable file format of the target OS. The Windows executable file is known as the Portable Executable (PE). This PE file does not only apply to executable files but also to DLL files and kernel mode drivers found on the target host. It is safe to say that the PE file consists of two most common file extensions that are an executable (.exe) and library file (.dll). A module is also referred to as a PE file, in cases of module it will imply that only one executable file will be part of a program or application.

The PE file consists of five main sections as indicated in figure 9 below:

- DOS MZ header
- DOS stub
- PE header
- Section table
- Sections

The PE file format contains a header followed by a series of sections. At a minimum the PE file will have two sections, one for code and the other for data. These sections contain either code or data and sometimes a combination of both. Some sections may contain data or code declared by the actual application, whereas other data sections contain important information for the operating system. The PE header starts with the signature of the file and contains the file properties, such as the number of sections and timestamp. The PE header describes vital pieces of the portable executable; it instructs the operating system on how to map the executable in memory.

MS DOS MZ Header
MS-DOS Stub Program
PE File Signature
PE File Header
PE File Optional Header
.text Section Header
.data Section Header
.rsrc Section Header
:
:
.text Section
.data Section
.rsrc Section
:
:
Debug Information

Figure 9: View of the PE File Format (Shaban, 2013)

According to (Shaban, 2013, Idika and Mathur, 2007, Marak, 2015, Elisan, 2015), the following are the most common and interesting sections in a PE file:

- **.text**: This section is the default code section that contains the instructions that the CPU executes. All other sections store data and supporting information. Normally, this is the only section that executes, and it should be the only section that includes code
- **.rdata**: This section contains the import and export information. It can also store other read-only data that is used by the application. Sometimes a file may contain .idata and .edata sections that stores the import and export information
- **.data**: The program's global data stores in this section that is accessible from anywhere in the application. This section does not store local data, or anywhere else in the PE file. It also contains Original Entry Point (OEP) of the file that refers to the execution entry point where the file execution commences of a PE file.
- **.rsrc**: The resources used by the executable including this section that are not reflected as being part of the executable such as images, strings icons, and menus. Strings can be stored either in the .rsrc section or in the main program, but they are often stored in the .rsrc section for Multilanguage support.

Table 3 lists the most common PE File Section

Executable Section	Section Description
.text	Contains the executable code
.rdata	Holds read-only data that is globally accessible within the software
.data	Stores global data accessed throughout the software
.idata	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in the .rdata section
.edata	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in the .rdata section
.pdata	Present only in 64-bit executables and stores exception-handling Information.
.rsrc	Stores resources needed by the executable
.reloc	Contains information for relocation of library files

Table 3: Sections of a PE File for a Windows Executable (Shaban, 2013, Idika and Mathur, 2007, Marak, 2015, Elisan, 2015)

### 2.6.2.1.1 Detailed Overview of the PE File Format

Each and every PE file will always start with the DOS-MZ header section that is located at the offset 0 of the PE file which will contain a pointer to the PE header section of the file. The DOS-MZ header is placed on top to enable a DOS OS to be able to identify the PE file as a valid executable file to be able to execute the DOS stub.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .L...J...yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	00	.....!

Figure 10: DOS-MZ Header - 1123211-090SD.exe (Section 4.2.2 Analysis of 1123211-090SD.exe)

The MS-DOS stub section is included for legacy applications and it is a valid DOS executable file. This section only informs the user that the file will not run in DOS mode when an attempt is made by the user to try to run the DOS mode as indicated in figure 11 below.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .L...J...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..... .....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	.....!
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	å ¢. 'í! , Lí!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program cannot
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$......
00000080	50	45	00	00	4C	01	03	00	74	C6	D9	57	00	00	00	00	PE..I .tÆUW....
00000090	00	00	00	00	E0	00	02	01	0B	01	08	00	00	10	07	00	.....à. ¢ ¢.+.•.
000000A0	00	20	00	00	00	00	00	00	3E	2D	07	00	00	20	00	00	.....>-•.....

Figure 11: PE Section Header - 1123211-090SD.exe (Section 4.2.2 Analysis of 1123211-090SD.exe)

The PE header also contains important information that is required to run the executable, for example:

- the base address of the PE File
- the address of the entry point
- the number of sections in the section table

This structure also contains important fields that the PE loader requires. The PE header is not located at the beginning of the file. The PE header is found in the offset 0x3C as indicated in figure 11 above. The four-byte value starting at address 0x3C is represented by the address of the PE header in relation to the start of the file. When the PE file is executed, the PE loader would go directly to the PE header. The PE loader would then bypass the DOS MZ header, DOS stub that then proceeds directly to the PE header (Elisan, 2015).

The section table is found between the PE header and the PE file's section and it is split into separate sections to store the contents of the file. The executable code is normally stored in the code section namely .text section as indicated in Figure 12.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00072000	00002000	00071000	00001000	00000000	00000000	0000	0000	60000020
.rsrc	00002000	00074000	00001000	00072000	00000000	00000000	0000	0000	40000040
.reloc	0000000C	00076000	00001000	00073000	00000000	00000000	0000	0000	42000040

Figure 12: Section Headers

Program data is stored inside the data (.data) section that is also the .text section. This section contains strings that assist in performing reverse engineering. As an example this section can contain network information such as hostnames and IP addresses. The last

section within the PE file is the sections with directory information that consists of up to 16 directories. Typically a fewer directories are found inside the PE file as the following:

- Import / Export Table
- Debugging
- Import Address Table

A program is known to contain several components that are assembled together. As soon as the executable file is created by the compiler, it is then linked with external functions or other executable objects using the *linker* as indicated in figure 13 below (Saffaf, 2009).

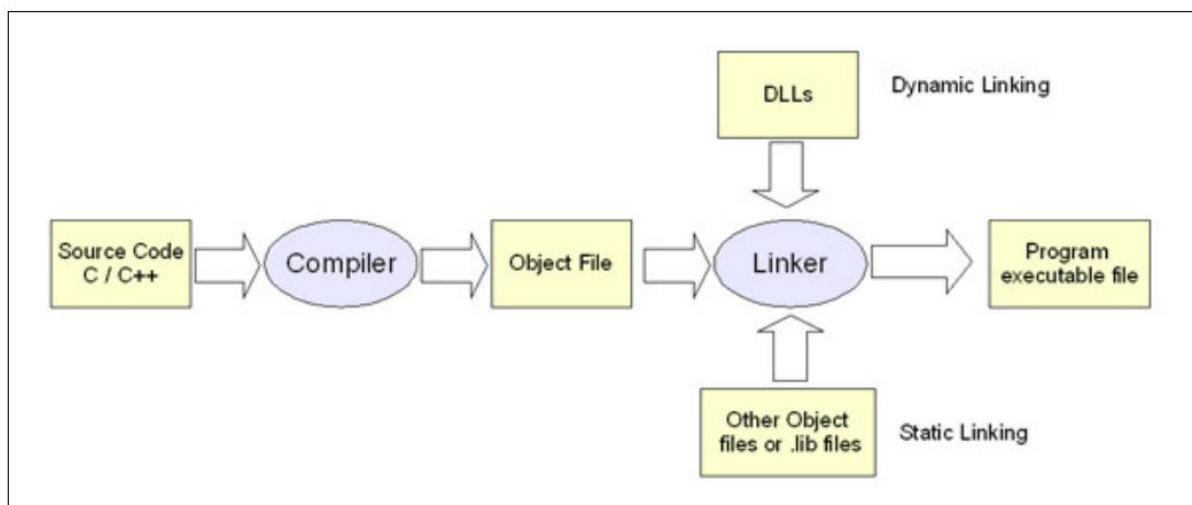


Figure 13: Linker creation of executable file (Saffaf, 2009)

The executable file imports function from the Win 32 API during runtime and this is achieved by using the DLL function. DLL is a library that can code and contains data that is used by more than one program at the exact period. This can help to promote the reuse of code and efficient usage of memory. Microsoft Windows, the kernel32.dll, user32.dll, gdi32.dll and ntdll.dll are native APIs presented to the programmer as DLL functions. These DLLs are also shared with other processes that can load or use them. The DLL is said to have a few drawbacks that can include:

- A Program using DLL that is dependent on the DLL
- and cannot be executed if this dependency is destroyed because of the DLL files that have been removed from the system. Programs that are using DLL are often known as dynamic executables as oppose to static executable applications

Static executables, which have a file extension of .lib, are embedded inside the application when it is developed. This will make the application independent of any shared libraries.

In dynamic executable the OS of has two different ways of linking the DLL during runtime that is known as load time and runtime linking. Load-time linking is the linking method whereby the linker would create a list in the PE file for the functions that the program can externally import and this list is called the import table. When the system loads the executable file, it then uses the information found in the import table to load all the DLL files that are used by the running executable. This also allows resolving external references made to run the program. Otherwise, the Runtime linking can load the DLL files and then import the required functions manually during runtime (Saffaf, 2009).

There are no import tables provided; as an alternative the executable imports the correct functions by loading the DLL file by first using Win32 AP's LoadLibrary function that is then followed by Win32 API's GetProcAddress function to obtain the address of the DLL's function required by the executable. Import table plays a vital role in dynamic linking function. It consists of a list of functions that the running executable can import that is grouped under each DLL linked with the running executable, as indicated in figure 14 (Saffaf, 2009).

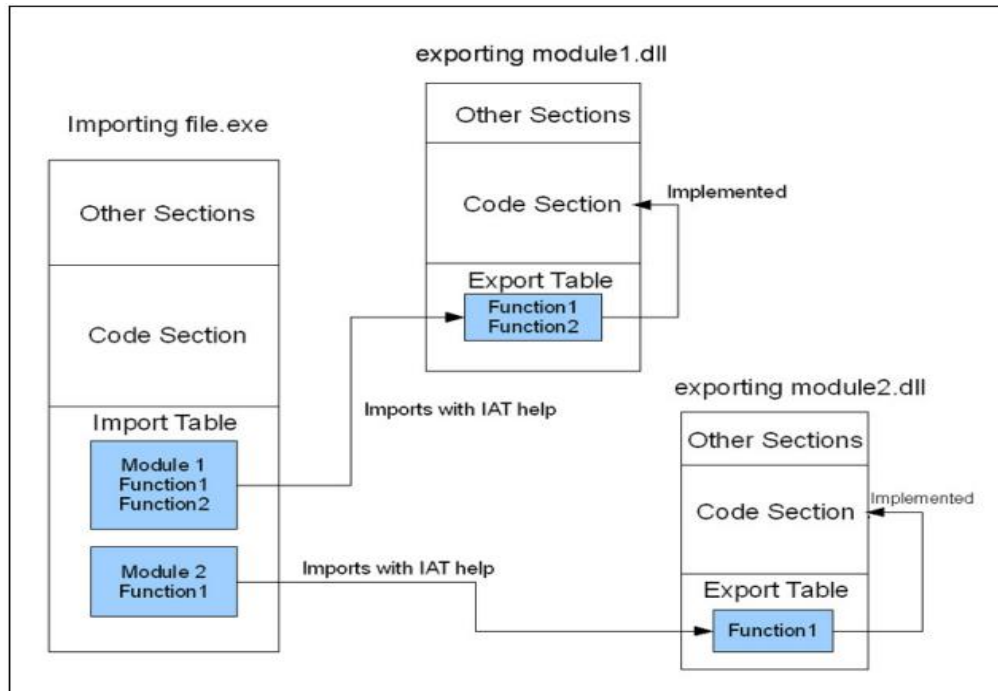


Figure 14: Imported and Exported Functions (Saffaf, 2009)

When modules or DLL provide a set of functions to other modules, these functions are then listed in the export table of that module. The export table contains the names and relative



After looking at the four examples above, we can simply conclude that the more randomness a file or executable retains, the higher the value of entropy. Entropy is used in different ways by malware authors and analysts; this provides a rough estimation of whether the file is encrypted or not and it helps to determine if further analysis is required.

### 2.6.2.3 Packer

**Packer Definition:** *Packing is a type of obfuscation that prevents the malicious code to be viewed until it is in memory which is achieved by using programs known as packers (Sikorski, 2012).*

A packed malware is malware that has been encrypted and compressed by real-time packers. Malware authors often use packers that are used to compress portable executable because it helps to hide the malware from being detected by antivirus technologies; most of these packers are easy to use and are available for free (Sikorski, 2012). The packer program takes the original file of the malware and compresses the file that then makes the original malicious code unreadable.

Packers are used on executables for two reasons viz.:

- a) To compress programs
- b) Thwart analysis or detection of malware

Malware can be found in a packed file and if malware is packed it cannot infect any system. Packers were created for legitimate reasons that were to decrease the overall file size of an executable or file. Malware authors started to utilize these programs as they began to realize the benefits as aforementioned.

According to (Osaghae, 2015, Shaban, 2013), packers can be classified based on their purposes and behaviours into four categories:

- **Compressors:** Simply shrink file sizes through compression with little or no anti-unpacking tricks. Popular Examples of compressors include the Ultimate PE Packer UPack, Ultimate Packer for Executables (UPX)
- **Crypters:** The Packer encrypts and obfuscates the original file contents and prevents the files from being unpacked without any compression
- **Protectors:** A hybrid packer that combines features from both compressors and crypters

- **Bundlers pack:** The Packer packages several executable and data files into a single bundled executable file that it then unpacks and accesses files within the package without extracting them to disk

A packed file may contain malware; until the antivirus solution knows how to unpack the file, the malware will not be detected. On the bright side, if the malware is packed it cannot infect the targeted computer. That would be the end of the story, except that we run-time packers; here is how they work. The packed file is an executable (program) that is only partially packed. A portion of the application is not packed. The start of the application is not packed, so when the packed executable is executed it starts unpacking the rest of the file. The un-packer tool is built right in.

Figure 16, below shows the malware sample being packed and this figure is part of the live experiments conducted as part of the research described in section 4.2.2 Analysis of 1123211 – 090SD.exe.

```
UPX Output Results
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ upx '1123211-090SD.exe'
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2013
UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

  File size      Ratio      Format      Name
-----
upx: 1123211-090SD.exe: CantPackException: .NET files (win32/.net) are not yet supported

Packed 1 file: 0 ok, 1 error.
```

Figure 16: UPX Output Results (Section 4.2.2 Analysis of 1123211 – 090SD.exe)

### 2.6.2.3.1 How does a packer work?

It works by using the parses Portable Executable (PE) internal structure. It rearranges the PE headers, sections, import tables, and export tables into new structures; the packed executable is compressed, encrypted and attaches a code segment that the malware will invoke before the OEP. Packers have the capability to pack the entire executable, including all data and the resource section, or pack only the code and data sections. The outer layer works to unpack the inner executable in memory and transfers execution to it.

Figure 17 shows its packed counterpart. This code is called the stub, and it decompresses the original data and locates the OEP. With the packed software, the unpacking stub is loaded by the OS, and then the unpacking stub loads the original software (Osaghae, 2015, Shaban, 2013, Sikorski, 2012). The packing software then performs three additional actions (Sikorski, 2012):

- a) It unpacks the original executable file into memory
- b) It decides all of the imports of the original executable file
- c) Transfers execution to the OEP

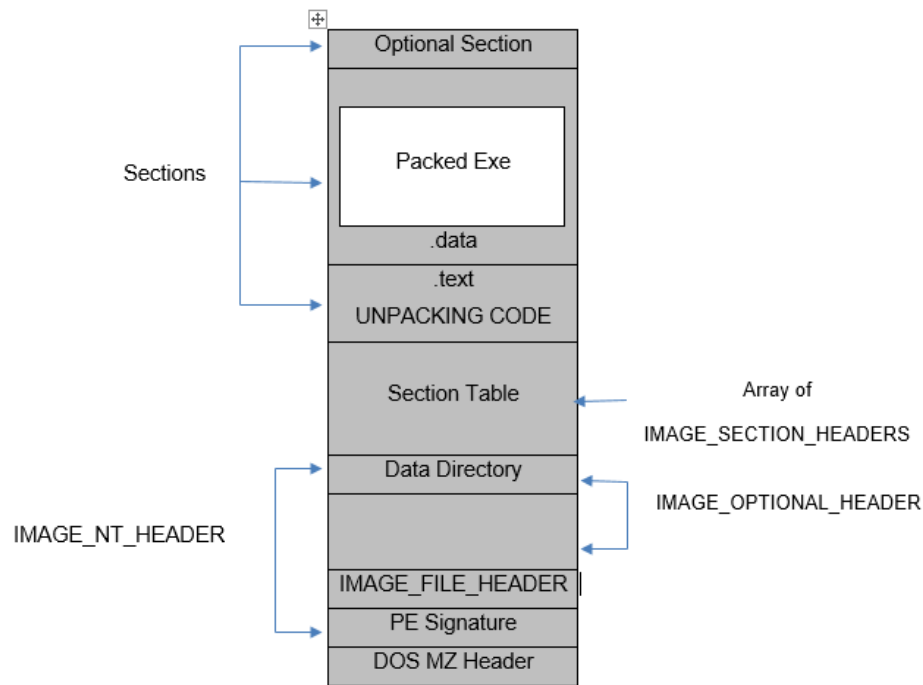


Figure 17: PE structure of a packed executable (Pietrek, 1994)

To be able to detect packers is a very crucial process in performing analysis of the acquired malware sample. If the packer is determined, we will then be able to unpack the code and then analyze the acquired malware. The packer detection method can be classified into static or dynamic analysis, based on where the detection takes place, either before the code is loaded or after into the memory. According to (Shaban, 2013), there are numerous heuristic IOCs that the file is packed, such as the following (Ligh et al., 2010):

- Small number of functions - The software file has only a few built-in functions, while normal, unpacked software will have many more
- Small number of imports - The software or file imports fewer than ten API functions from libraries supplied by the OS. This shows that either the file is very limited in functionality or a packer has “hidden” the API functions
- Encoding instructions inside a loop - The series of IMUL, ADD, SHR, XOR, and AND instructions with hard-coded and the numbers inside a loop shows that the application carries out obfuscation or de-obfuscation of some type

LoadLibrary and GetProcAddress are two additional functions, which are used to load and gain access, that are frequently included on packed and obfuscated code (Sikorski, 2012).

#### 2.6.2.4 Obfuscation

Definition of Obfuscation according to Joshua Cannel (Cannell, 2013):

“**Obfuscation**: is a technique that makes binary and textual data unreadable and/or hard to understand. Software developers sometimes employ obfuscation techniques because they don't want their programs being reverse-engineered or pirated.”

##### **Malware Obfuscation definition explained**

Today's technology is not able to detect malware artifacts as these technologies often rely on common properties and ignore the semantics of malicious executable code within the malware artifact and this is due to the malware code being obfuscated.

The obfuscated code within the malware artifact ensures that the malware stays hidden from detection while infecting the system, which makes it then prevent the malware from being removed or analyzed (Cannell, 2013).

Malware achieves this by using techniques to prevent detection and analysis such as obscuring the filenames, changing the attributes of the file and hide and operate alongside legitimate applications or services (Cannell, 2013).

##### **2.6.2.4.1 Common methods of code obfuscation**

There are four types of obfuscated malware that include the following:

- a) **Encryption**: This method of evasion is used to hide the presence of malware by encrypting the payload of the malware. The encrypted malware type consists of a decryptor that includes a key for encryption and the payload that is encrypted. The encrypted malware makes it difficult for the malware detectors to detect the malicious payload (Schiffman, 2010)
- b) **Stealth**: The malware would normally hide its true intention it makes on the infected system by only showing unmodified data or changes to the malware detectors. For example, when an antivirus solution scans through the areas that are possibly infected on the system, the malware would indicate a clear state of the system that does not reflect any infections
- c) **Oligomorphic**: This type of malware uses evasion method to encrypt its payload the same way as the encrypted malware and the difference with this type of malware is that the decryptor is changed by the malware to replicate itself. While making use of the decryptor



as the signature, the oligomorphic malware makes it difficult to be detected which may require additional attention by the decryptor generator of malware (Szor, 2005)

- d) **Polymorphic:** This type of malware is armed with the same evasion methods as the other aforementioned malware, however this malware consists of body that is encrypted with various copies of the malware decryptor. This malware creates occurrences that use diverse keys of encryption in various occurrences so that each occurrence of the malware body appears different from other variants of the malware (Schiffman, 2010)
- e) **Metamorphic:** Malware authors have added obfuscated code to the entire body of the malware that also incorporates the aforementioned evasion methods. The difference with this type of malware is that it does not make use of a malware decryptor and a consistence code; instead it then makes use of a single application code that replicates into totally different variants of malicious code (Szor, 2005)

#### **2.6.2.4.2 History and Examples of Obfuscated code**

The first malware that was created in 1986 by Farooq Alive brothers, was called the Brain malware, this malware attempted to hide its existence (Schiffman, 2010). This malware would attempt to cover-up the sectors of the hard disk sectors that it had infected and would only display un-modified data on the disk.

Another malware artifact, which was called the Cascade malware that was also discovered late in the 1986s used encryption to scramble its code or contents. This malware contained encryption or decryption routines which followed the body of the encrypted viral code that was a technique that was eventually adopted by every malware that was encrypted (Schiffman, 2010). The Cascade malware made use of symmetrical XOR cipher.

The XOR cipher was the obvious choice during those times because of two reasons (Schiffman, 2010):

- a) Antivirus solutions were based on pattern matching and were not able to scan encrypted malware as the malware body contained random data of bytes. The XOR of encryption or decryption routine, which preceded the actual malware would only be the detectable pattern known as the decryptor. The problem was also that the antivirus solution was not able to differentiate between strains of the same malware nor identify dissimilar malware that shared the same cryptographic routines.
- b) Meanwhile the operation of XOR is also symmetrical and alterable; it offered malware authors the ability and quickness to have functions to encrypt and decrypt. In order to explain the XOR operation it is important to underline the functions of XOR in detail

## XOR operation explained

XOR Definition (Sastry and Kumar, 2012): **Exclusive** or Exclusive disjunction is a logical operation that outputs true only when inputs differ (one is true, the other is false).

The XOR operation consists of two inputs and one output. It is like the ADD operation that takes two arguments (two inputs) and produces one result (one output). The inputs to an XOR operation binary can only be 0 or 1 and the result can only be 0 or 1. The binary XOR operation which is also known as the binary XOR function, will always produce a 1 output if either of its inputs is 1 and will produce a 0 output if both of its inputs are 0 or 1 (Systems).

If we call the inputs A and B and the output C we can show the XOR function as shown from the example below (Systems):

A		B		C
0	XOR	0	->	0
0	XOR	1	->	1
1	XOR	0	->	1
1	XOR	1	->	0

The machine code XOR instruction operates on 8 sets of inputs and outputs in parallel.

If we have XORed two input bytes together then we get an output byte. If we give each bit within a byte a number we can see that each bit in the output is the result of the XOR function on two corresponding bits of the input that is (Systems):

A7	A6	A5	A4	A3	A2	A1	A0
B7	B6	B5	B4	B3	B2	B1	B0
C7	C6	C5	C4	C3	C2	C1	C0

If there are two binary numbers 00100100 and 00100001 we can see the effect of XORing these two sets of 8 bits in parallel (Systems):

### Example

Argument - 1	0	0	1	0	0	1	0	0
Argument - 2	0	0	1	0	0	0	0	1
<b>Result =</b>	0	0	0	0	1	0	1	1

- i. The input bits A5 and B5 and the output bit C5 are here shown in red
- ii. The input bits A2 and B2 and the output bit C2 are here shown in green

- iii. The input bits A1 and B1 and the output bit C1 are here shown in yellow
- iv. The input bits A0 and B0 and the output bit C0 are here shown in blue

In the XCSB of the XOR binary operator it will work the same way; operating in parallel on the sets of inputs and outputs within a variable or constant. If the value 0x24 is assigned to the variable J, which is the hexadecimal that is equal to the binary value of 00100100, and the value 0x21 to the variable K, which is the hexadecimal equivalent of the binary value 00100001, we then perform the XCSB XOR operation on J and K and assign the result to M. The value stored in M will be 0x05 that will be the hexadecimal equivalent of the binary value 00000101.

Written as XCSB source code this would be:

```
J = 0x24
K = 0x21
M = J ^ K
```

Example of XOR operation (Cannell, 2013)

The XOR operation is the most common method for malware code obfuscation as this is the easiest code to implement and to hide malware code for analysis. Figure 18 below indicates the data highlighted, which is obfuscated data, and it is unreadable in the current form.

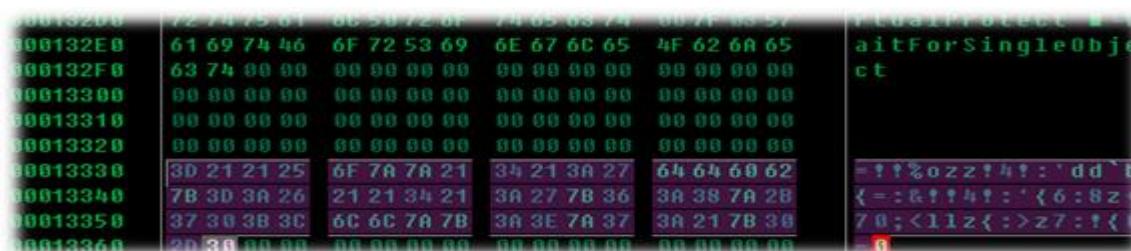


Figure 18: XOR operation-encoder

In the current unreadable form of the data the XOR value of 0x55 is applied that then reveals a URL, which could be concluded to be a malicious URL, as indicated in figure 19 below.

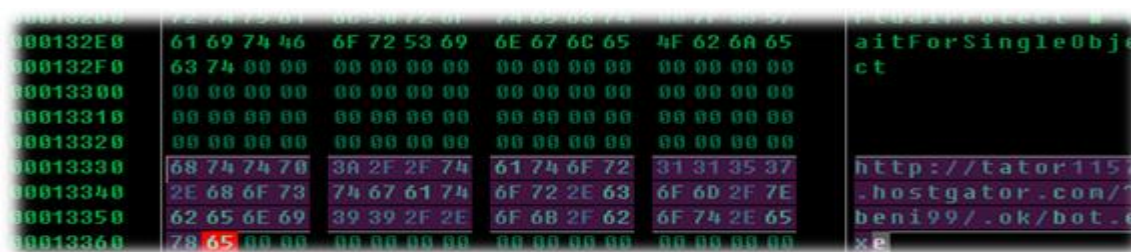


Figure 19: XOR operation – decoder

The URL indicated in figure 19 above indicates that the malware will try to contact this URL which is “http://tator1157.hostgator.com”, that will then download a malicious “bot.exe” to

perform further attacks on the infected system (Cannell, 2013). The above form of obfuscation can be defeated by using applications that can manually cycle through every XOR byte value searching for particular malicious strings. One of the most common applications which can be executed on a Linux or Windows operation system is called the XOR Search application. The XOR search application was written by Didier Stevens and this application searches for strings that are encoded in multiple formats, also including XOR.

Malware authors are aware that such applications exist; thus they implement further obfuscation techniques to avoid detection by employing a multi-cycle method. This will then perform an XOR operation against the data with a particular value; thus making the second value to pass through another value.

Example of Common String and Payload Obfuscation Techniques in Malware (Howard, 2010):  
This example based on the malware with the following hash SHA-256, is

*f4d9660502220c22e367e084c7f5647c21ad4821d8c41ce68e1ac89975175051.*

The malware artifact indicates some of the most common methods used by malware authors to complicate malware analysis of malware using dynamic and static analysis. The malware creates vectors that contain lists of the following applications from a dynamic analysis of the malware:

- a) "OLLYDBG"
- b) "W32DASM"
- c) "WIRESHARK"
- d) "SOFTICE"
- e) "PROCESS EXPLORER"
- f) "PROCESS MONITOR"
- g) "PROCESS HACKER"

All windows on the infected system are enumerated by this malware. If any of the windows are found to belong to any of the applications listed above, the malware will execute further by entering into a loop until the applications are closed. This is a basic method that is easy to deploy or implement in a malicious code of the malware.

When static analysis is performed on the same malware, the malware then employs two additional methods that include:

- a) Strings in the malware are encrypted using a custom encryption scheme that have the following implications for the malware authors and analysts:

- ✓ C&C domain(s) are hard-coded in the malware that requires malware authors to generate DGA's. API's used by the malware resolve when the malware is executed and the names of those APIs are also decrypted. This then means that static analysis becomes more explanatory after the encryption scheme is understood by the malware analyst
- b) Communications to the C&C is often encrypted with custom schemes:
- ✓ Communication of malware with the C&C using custom-encrypted or obfuscated communication protocol on top of regular HTTP. This will then allow a generic module that provides a low-cost solution for changing the communication scheme between the infected clients and the C&C to be generated by malware authors to generate. This can additionally render real-time network analysis or monitoring to be partially or totally ineffective.

Quote by Mike Schiffman (Schiffman, 2010)

“Obfuscation of malware serves the one ultimate purpose: *Survival.*”

#### **2.6.2.5 OLE2**

**Object Linking and Embedding (OLE2)** is defined as a proprietary technology developed by Microsoft that allows embedding and linking to documents and other objects. For developers, it brought OLE Control Extension (OCX) that is a way to develop and also to use custom elements of the user interface. On a technical level, an OLE object is any object that implements the OLE object interface and possibly along with a wide range of other object interfaces, depending on the needs of the object's (Tool, 2016).

OLE is used in other file formats as its underlying container file. It allows data to be stored in multiple streams of data. The OLE is also known as:

- Compound Binary File
- Compound Document File
- OLE2 file

#### **2.6.2.6 VBA Macros**

The VBA project automatically runs if the user enables macros once when opening the file downloaded from the internet. There are office targeting macro based malware that execute macros when enabled on the targeted system. There are two versions of the malicious macros that are categorized further as the following:

- Old macros
- New macros

The old macros only decrypted the older version of the malware that appends to the document and executes it. In this example, the malware carry the bait document itself that will be responsible to open and delete the original document. In new macros, the macro would decrypt and start the bait document and the malware; both will be appended to the document. The malicious macro would be implemented as a VBA code and would use the predefined names Auto-Open an Auto-close to execute the functions automatically after the malicious document is opened and closed.

## **2.7 Malware Analysis Lab**

Before malware can be analysed, we need to understand the tools that will be used, and why it is important to understand how to design a secure and isolated environment in which malware can be analysed. This chapter also proposes a methodology of designing and implementing an environment in which malware analysis will be performed as part of the research.

When malware is analysed and executed in an isolated environment, the malware exposes a malicious behaviour; the environment does not allow the malware to execute beyond the isolated environment (Rossow, 2013). The sandbox environments and tools which are proposed in this chapter are used to conduct experiments as part of the research.

### **2.7.1 Architecture Overview**

The malware analysis environment consisted of virtual machines (VM) configured with a virtual network to allow all the VM's to access. Sandbox VM's are used to conduct experiments to fulfil the requirements of this research. The malware analysis environment is built to be scalable. Sandbox analysis will be used to perform malware analysis in an isolated environment; sandbox analysis is also referred to as black-box analysis that is a method that is used to analyse malware characteristics and behaviours by executing the malware sample in an isolated controlled environment. The behaviour of the malware was monitored, such as registry activity, system and file activity and network activity (Kasama, 2014). There are two types of sandboxes configured as part of this research, viz. Cuckoo and REMnux sandboxes respectively. Both sandbox environments are discussed in detail later in this chapter.

#### **2.7.1.1 Windows 7 Virtual Environment Setup Overview**

A Windows 7 with service pack 1 VM was installed and configured within the Virtualbox software. The system was also set up on the same virtual network with the REMnux system discussed in section 5.1.3 below. Once the system was installed, static analysis tools were installed, such as CFF explore and PEStudio.

### **2.7.1.2 Cuckoo Sandbox Environment Setup Overview**

The sandbox environment consisted of a Linux VM which was installed with an Ubuntu operating system that acted as gateways between the windows VM and the online resources. Normal operation of all network traffic from the Windows VM was prohibited from accessing the Internet and the traffic was re-routed to network simulator (INetSim) configured on the Sandbox VM.

The sandbox VM network was configured as a virtual network and only the VM environment was able to access that network and all network traffic that was re-routed through a virtual network and sent to the relevant VM sandbox. The Linux VM was used to perform automatic analysis of malware. During the analyses, all network traffic was captured and checked against intrusion detection system (IDS), configured on the Linux VM.

The Windows VM was installed with a Windows 7 (32-bit) operating system where malware samples or artifacts are executed and analysed. Regular snapshots of the VM were taken before any malware was analysed. The snapshot was then used for automatic analyses within Sandbox and consisted only of minimal toolsets.

### **2.7.1.3 REMnux Malware Analysis Environment Overview**

The REMnux system ran on a very light weight version of Ubuntu operating system and the system was installed with various tools that were used to perform static and dynamic analysis of malware. The REMnux system was configured to virtually communicate with a Windows VM and the traffic between the two systems was isolated from the rest of the environment.

Regular snapshots were taken on the Windows VM that was used during manual or static analysis process. The Windows VM was also installed with minimum tools required to perform malware analysis that include but are not limited to the following:

- debuggers
- disassemblers
- hex editors
- portable executable (PE) viewers

The windows VM was only used for further analysis but most of the static analysis experiments were conducted from the REMnux system as it already had the required toolsets required to perform static analysis of the malware samples or artifacts.

## 2.7.2 Physical and Virtual Environments

### 2.7.2.1 Physical Machines Environment (Syarif Yusirwan et al., 2015)

The use of physical machines consists of advantages and disadvantages described in Table 4 below, when it comes to having a secure malware analysis environment.

Table 4: Physical Machines Environment

Advantages	Disadvantages
Provides a structure of a scenario that is as close to the real world as possible.	Analyzing and executing malware requires a significant amount of resources that must be set up and it is also a lot more difficult to maintain. It is also an expensive option to opt for to set up a malware analysis environment.
It allows malware to be executed and analyzed to reflect ideal or real world conditions as closely as possible.	Setting a number of machines running with different types of Operating systems and to roll back the machines back to the original state takes a lot more effort and it is also time consuming. Additional tools will then be required to restore the machine to original state such as system imaging software.
It will allow malware to execute free with less effort and will not require anti-malware detection measures to be implemented.	It will be difficult to set up multiple interconnected systems; this will require a data center on its own.

### 2.7.2.2 Virtual Machine Environment (Syarif Yusirwan et al., 2015)

The use of a virtual environments and machines consists of advantages and disadvantages described in Table 5 below, when it comes to having a secure malware analysis environment.

Table 5: Virtual Machine Environment



Advantages	Disadvantages
Installing and setting up a virtual environment requires less effort and will only require virtual environment tools such as Oracle Virtualbox. This is a cost effective solution to implement for setting up a malware analysis environment.	Anti-malware counter measures need to be implemented in order to ensure that the malware executes to simulate a physical environment as malware can detect that it is executing in a VM.
The virtual machine (VM) tools allow for scaling up and adding resources and VMs.	Malware can exploit vulnerabilities in the VM software itself and this can increase the risk of infecting the host machine and end up infecting the other machines on the network.
It also requires less effort to configure VM snapshots in order to restore the VMs to the original state.	Regular updates of VM software is required in order to mitigate VM vulnerabilities.

### 2.7.3 Malware Analysis Isolated Environment

The malware analysis environment is scoped and designed based on the malware analysis techniques that will be performed during the malware analysis process. The two techniques for performing malware analysis, which have been already discussed in the previous chapters of the research, are:

- a) Static analysis
- b) Dynamic analysis

These two techniques are very different but are both essential for performing a thorough analysis of malware in an isolated environment.

#### 2.7.3.1 Isolated Environment Design

The process of setting up an isolated environment is a critical step that requires proper planning, resources and understanding the requirements needed to perform malware analysis. There are two possible ways of setting up an isolated environment that can be using physical dedicated machines or using virtual machines (VM)(Syarif Yusirwan et al., 2015). For the purpose of this research VMs were designed and implemented as the malware analysis environment requires significant amounts of resources.

### 2.7.3.2 Virtualized Environment

For the purpose of this research it is important to first understand what a virtual machine is, the technology and why it is important for the purpose of our research. A virtualised environment is an environment that runs as an application that allows for one or more operating system to be run as if they are installed on a dedicated hardware (Sanabira, 2007) and this environment is called a virtual machine (Virtualbox, 2016). This environment allows a single computer to share its resources among multiple systems running simultaneously (Michael A. Davis, 2010). This environment, called a "virtual machine", is created by virtualization software by intercepting access to limited hardware components and features. The physical system is known as a "host" and the virtual machine is known as a "guest". Most of the guest code or OS runs directly on the host computer, and the guest OS knows it's running on real machine (Virtualbox, 2016).

The concept of an Operating System running within an Operating system (OS) creates for more terms:

- Host OS – The OS that runs the VM environment
- Guest OS – The OS that runs within the VM environment host

There are several commercial and free virtualization software available, viz. a free virtualization software called Virtualbox that was used for the purpose of this research. Figure 20 below demonstrates a Windows 7 operating systems running on a Virtualbox software.

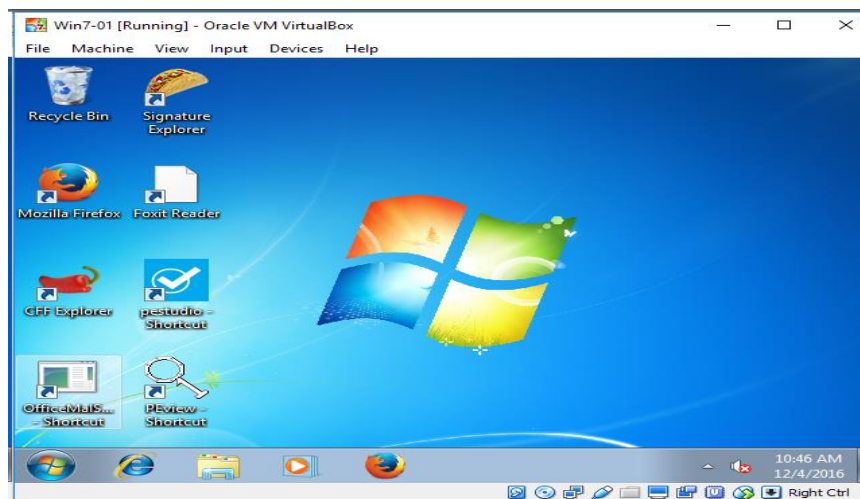


Figure 20: Virtual Box Software running Windows 7 System

According to the author of the Hacking Exposed Malware & Botnet solutions book, there are two types of virtual machines (Davis et al., 2010):

Types of Virtual Machines:

- The process virtual machine that is known as an application virtual machine and is normally installed on an OS that can virtually support a single process. This type of VM provides an execution environment that is also known as a sandbox environment for running a process that can use and manage resources on the system on behalf of the process
- Hardware virtual machines provide low-level hardware emulation for multiple operating systems, known as guest operating systems, to use simultaneously. This means the VM mimics x86 architecture, providing all of the expected hardware and assembly instructions. This emulation or virtualization can be implemented in “bare-metal” hardware (meaning on the CPU chip) or in software on top of an existing running operating system known as the host operating system. The operator of this emulation is known as the hypervisor (or virtual machine manager, VMM) (Davis et al., 2010).

### **2.7.3.3 Operating System Consideration**

Malware is designed to behave and execute differently depending on the operating system that is analysed. Some malware is maybe designed to function on a Windows based Operating system environment and other malware may execute on a Linux operating environment. Malware that is executed on a Windows operating system might be designed to connect with a command and control (C&C) server (Svajcer, 2015). It is important to set up and configure different types of operating systems such as Linux and Windows systems when performing malware analysis. These are the systems that will be used for analysing malware in an isolated environment.

### **2.7.3.4 Virtual Network Consideration**

Malware analysis hosts must be analysed in an isolated environment from the live or production network. This is due to other types of malware that have the ability to replicate themselves onto other systems such as worms or infect other systems in order to connect with C&C and be able to use the infected hosts as distributed botnet systems, or even use the infected hosts to perform further malicious activities (Elisan, 2015). The malware analysis environment can be isolated even from reaching the internet or online resources (Svajcer, 2015). With all the considerations taken into account the malware analysis environment must be completely isolated from the rest of the network, even the host machine.

## **2.7.4 Malware Analysis System Sandboxes Overview**

### **2.7.4.1 Sandbox Systems Descriptions**

Sandbox technology is configured and Anti-virus engine is used that include the follow sandboxes:

- Windows 7 System

- REMnux
- Cuckoo
- Virus Total

### 2.7.4.2 Windows 7

Figure 20 above is the Windows 7 system that was configured on the same virtual network as the REMnux system. Various static analysis tools were also installed as indicated on the desktop of the above figure 20.

### 2.7.4.3 REMnux

REMnux is a free Linux toolkit for assisting malware analysts with reverse-engineering malicious software. It strives to make it easier for forensic investigators and incident responders to start using a range of freely available analysis tools that can dissect malware, yet could be hard to locate or set up (Zeltser, 2016). The toolkit runs on an Ubuntu operating system and consists of several tools that can be used to analyse or reverse engineer malware in Windows or Linux operating systems. Figure 21 and 22 demonstrate a few useful tools built within the REMnux system.

```

remnux@remnux:~$ pescanner.py kroker.exe
#####
[0] File: kroker.exe
#####
Meta-data
=====
Size      : 172932 bytes
Type     : PE32 executable (GUI) Intel 80386, for MS Windows
Architecture : 32 Bits binary
MD5      : 7d5f648df659bb98454d1613b4f9444a
SHA1     : 6c312a3f8f3dde918dda934a47c4b0e95c6884c
ssdeep   : 3072:YBntKcw018Y0ISRgINHMcPRkdZAdCO+14gFG2h51B2y7Uqz1Q7:Lfu+EHgIFR0eAAV1B0YDEr
imphash  : f6df5ac4562491c509e59c150de970bf
Date     : 0x4AAB9BE7 [Sat Sep 12 13:02:31 2009 UTC]
Language : []
CRC: (Claimed) : 0x0, (Actual): 0x345f5 [SUSPICIOUS]
Packers  : Xtreme-Protector v1.05
Entry Point : 0x4073a0 .text 0/5
=====
Offset | Instructions
-----|-----
0      | jmp 0x407e33
5      | add [eax],al
7      | add [eax],al
9      | add [eax],al
11     | add [eax],al
13     | add [eax],al
15     | add [eax],al
17     | add [eax],al
19     | add [eax],al

```

Figure 21: REMnux tools overview – PESCANNER (Westcott and Zeltser, 2016)

```

remnux@remnux:~$ inetsim
INetSim 1.2.5 (2014-05-24) by Matthias Eckert & Thomas Hungenberg
Using log directory: /var/log/inetsim/
Using data directory: /var/lib/inetsim/
Using report directory: /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 2818) ===
Session ID: 2818
Listening on: 192.168.187.167
Real Date/Time: 2015-06-13 08:41:58
Fake Date/Time: 2015-06-13 08:41:58 (Delta: 0 seconds)
Forking services...
 * ftps_990_tcp - started (PID 2827)
 * pop3_110_tcp - started (PID 2824)
 * smtp_25_tcp - started (PID 2822)
 * smtps_465_tcp - started (PID 2823)
 * pop3s_995_tcp - started (PID 2825)
 * ftp_21_tcp - started (PID 2826)
 * https_443_tcp - started (PID 2821)
 * http_80_tcp - started (PID 2826)
done.
Simulation running.

```

Figure 22: REMnux tools overview – INETSIM (Westcott and Zeltser, 2016)

#### 2.7.4.4 Cuckoo Sandbox

Cuckoo Sandbox is a free software that automated the tasks of analysing any malicious file that is executing in Windows, OS X, Linux and Android systems.

Cuckoo Sandbox is a malware analysis sandbox system; when any suspicious file is imported into it, in a matter of seconds the Cuckoo system will provide a detailed report of the results outlining what such a file did when it was executed inside an isolated environment that is running on operating system that is configured on the victim VM.

Malware is the swiss-army knife of many cybercriminals and any other adversary to your corporation or organization. In these changing times, detecting and removing malware artifacts is not enough: it's vitally important to know how they operate in order to understand the properties or context, the motivations and the goals behind the breach and for better protection the future (Claudio Guarnieri, 2013).

Figure 23, provides a graphical view of how the Cuckoo Sandbox operate. The Cuckoo Sandbox is run on an Ubuntu 16.0 64bit operating system. The sandbox has the following features, scripts, parsers and decision tree.

- The sandbox has a virtual guest operating system runs a Windows 7 operating system which is managed by a virtualbox application.
- When a malware sample is submitted on the front end of the Cuckoo sandbox, the malware sample is then sent to the guest Windows 7 system for analyses by the Cuckoo scripts.
- Once the file has been analyzed from the guest system, a report is generated by the Decision tree feature.

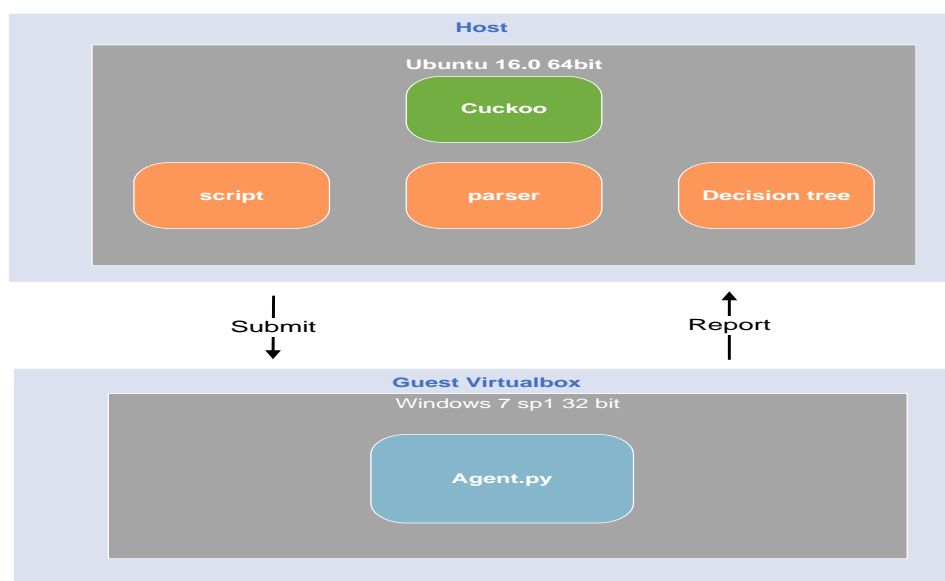


Figure 23: Cuckoo Sandbox Overview (Guarnieri et al., 2013)

### 2.7.4.5 Incident Response Malware Analysis (IRMA)

IRMA sandbox system is an open-source platform designed to help malware researchers and analysts to identify and analyse malicious files. Today's defence is not only about learning about a file, but it is also getting a fine overview of the incident that might occur or have occurred already, which provides the results of where / when a malicious sample has been seen, who submitted the hash, where the hash was noticed, which antivirus solution detects or can detect the malicious sample and so forth. Each malicious file submitted to the IRMA sandbox is then analysed in various ways (Quint et al., 2016).

Figure 24, this is the front end of the Incident Response Malware Analysis system that was used as part of the research. This system provided additional analysis on the behavior of the malware samples that were analyzed.

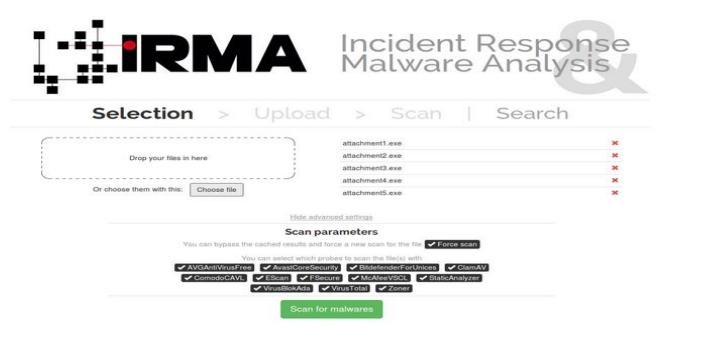


Figure 24: IRMA Web Interface Overview (Quint et al., 2016)

### 2.7.4.6 Virus Total

Virus Total is a free web-based and independent anti-virus service that makes use of multiple anti-virus engines to analyse suspicious files. The suspicious files can be uploaded to the web site that then gets to be analysed by over 52 anti-virus engines. According to Virus Total, it will scan, detect and if appropriate, any type of binary content, be it a Windows executable, Linux file, PDFs, images, java script code, etc. Most of the antivirus companies that use this site will have solutions for multiple platform; hence they usually produce detection signatures for any kind of malicious content and there is not any solution that offers a 100% effectiveness rate for detecting viruses and malware.

Figure 25, below provides a view of what the front end of Virus Total that was used as part of the research.



Figure 25: Virus Total Web Interface Overview (Total, 2016)

## 2.7.5 Tools based on type of analysis

Windows operating systems running Windows 7 were installed as VMs and have various malware analysis tools installed. These tools are classified into two categories that are behavioural and code analysis tools.

### 2.7.5.1 Code-Analysis / Static Tools (Zeltser, 2016)

Analyzing the code of the malicious file or software assists in determining the characteristics that might be difficult to obtain through the tools used in behavioural analysis. In the case of a malicious executable, you rarely will have the luxury of access to the source code from which it was created. The following tools are used to reverse compiled executables created to execute in Windows:

- **Disassembler and debugger:** OllyDbg and IDA Pro Freeware can parse compiled Windows executables and can act as disassemblers that can display their source code as assembly instructions. These two tools also have debugging capabilities that allow execution of the most interesting parts of the malicious application bit by bit and also under well-ordered conditions and allow a better understanding of the source code
- **Memory dumper:** OllyDumpEx helps to obtain protected code located in the infected system's memory and also dumps it to a text file. This approach is useful when analyzing packed executables that are difficult to disassemble because they encode or encrypt their instructions, extracting them into RAM only during run-time.

### Toolsets used from the REMnux and Windows 7 systems (Tool, 2016)

Table 6, below is a summary of all the tools that are built in on the REMnux systems and tools that were installed on the Windows 7 system implemented as part of this research study.

Tool Name	Description
File	This command or tool provides property details of the file
Clamscan	This tool is an open source antivirus engine for detecting trojans, viruses, malware & other malicious threats. Clamscan is a command line anti-virus scanner.
RHash	RHash (Recursive Hasher) is a console utility for computing and verifying hash sums of files. It supports CRC32, MD4, MD5, SHA1, SHA256, SHA512, etc.
Balbuzard.py	Balbuzard is a package of malware analysis tools in python to extract patterns from suspicious files (IP addresses, domain names, known file headers, interesting strings, etc.). It can also crack malware obfuscation such as XOR, ROL, etc. by brute forcing and checking for those patterns.
Oledump.py	It is a tool written by Didier Stevens and it is a program to analyze OLE files. These files contain streams of data. Oledump allows you to analyze these streams.
Officeparser.py	<p>Officeparser.py is a python script that parses the format of OLE compound documents used by Microsoft Office applications.</p> <p>Some useful features of this script include:</p> <ul style="list-style-type: none"> <li>• macro extraction</li> <li>• embedded file extraction</li> <li>• format analysis</li> </ul>
Olevba	Olevba is a script to parse OLE and OpenXML files such as MS Office documents (e.g. Word, Excel), to detect VBA Macros, extract their source code in clear text, decode malware obfuscation (Hex/Base64/StrReverse/Dridex) and detect security-related patterns such as auto-executable macros, suspicious VBA keywords used by malware, and potential IOCs (IP addresses, URLs, executable filenames, etc.).
Pescanner	<p>Scans the executable for suspicious characteristics and packer signatures. Pescanner.py is a PE analyzer written in python by the authors of the Malware Analysts Cookbook.</p> <p>The script has the ability to detect:</p> <ul style="list-style-type: none"> <li>• Files with TLS entries</li> <li>• Files with resource directories</li> <li>• Suspicious IAT entries</li> </ul>



	<ul style="list-style-type: none"> <li>• Suspicious entry point sections</li> <li>• Sections with zero-length raw sizes</li> <li>• Sections with extremely low or high entropy</li> <li>• Invalid timestamps</li> <li>• File version information</li> </ul>
Pedump	<p>A pure ruby implementation of win32 PE binary files dumper.</p> <p>Supported formats:</p> <p>DOS MZ EXE</p> <p>win16 NE</p> <p>win32 PE</p> <p>win64 PE</p> <p>Can dump:</p> <p>MZ/NE/PE Header</p> <p>DOS stub</p> <p>'Rich' Header</p> <p>Data Directory</p> <p>Sections</p> <p>Resources</p> <p>Strings</p> <p>Imports &amp; Exports</p> <p>VS_VERSIONINFO parsing</p> <p>PE Packer/Compiler detection</p>
PEframe	<p>PEframe is an open source tool to perform static analysis on Portable Executable malware and generic suspicious file. It can help malware researchers to detect packer, xor, digital signature, mutex, anti-debug, anti-virtual machine, suspicious sections and functions, and much more information about the suspicious files</p>
Pestr	<p>Searches for strings in PE files</p>
Exescan	<p>This tool is part of the pev PE file analysis framework, and its primary purpose is to extract strings from Windows executable files. However, this tool goes beyond the traditional strings tool by providing options to show the offset of a string within a file and the section where it resides</p>
OfficeMalScanner	<p>OfficeMalScanner is an MS Office forensic tool to scan for malicious traces, like shellcode heuristics, PE-files or</p>

	<p>embedded OLE streams. The tool will look for several strings and API calls to guess if the document is likely to be malicious, such as:</p> <ul style="list-style-type: none"> <li>• FS:[30h]</li> <li>• FS:[00h]</li> <li>• API-Hashing signature</li> <li>• API-Name GetSystemDirectory string</li> <li>• API-Name CloseHandle string</li> <li>• API-Name VirtualAlloc string</li> <li>• API-Name GetProcAddress string</li> <li>• API-Name LoadLibrary string</li> <li>• Function prolog signature</li> <li>• CALL next/POP signature</li> </ul>
CFF Explore	<p>The tool was created by Daniel Pistelli; it's a suite of tools including a PE editor called CFF Explorer and a process viewer. The PE editor has full support for PE32/64. Special fields description and modification (.NET supported), utilities, rebuilder, hex editor, import adder, signature scanner, signature manager, extension support, scripting, disassembler, dependency walker, etc.</p>
PE Studio	<p>Malicious software often attempts to hide its intents in order to evade early detection and static analysis. In doing so, it often leaves suspicious patterns, unexpected metadata, anomalies and other indicators.</p> <p>The goal of PEstudio is to spot these artifacts in order to ease and accelerate Malware Initial Assessment. The tool uses a powerful parser and a flexible set of configuration files that are used to detect different types of indicators and determine thresholds. Since the file being analyzed is never started, you can inspect unknown or malicious executable files, trojan and ransomware without a risk of infection.</p>

Table 6: Toolsets

### 2.7.5.2 Behavioural / Dynamic Analysis Tools (Zeltser, 2016)

- **File system and registry monitoring:** Process Monitor includes ProcDOT that offers a way to observe how to locate processes read, write, or registry entries, deletions and

files. These tools can help you understand how malware attempts to embed into the system during infection

- **Process monitoring:** The process explorer and also process hacker tools replace the built-in Windows Task Manager, helping you observe malicious processes, including local network ports that an attempt to open them, may take place
- **Network monitoring:** Tools such as Wireshark are used as a network sniffer that observes and monitors the network traffic for malicious communication attempts, such as botnet queries, DNS requests, and/or downloads
- **Change detection:** Regshot is a lightweight tool for comparing the system's state before and after the infection to highlight the key changes malware made to the file system and the registry

The above mentioned tools provide the analyst or research with a sense of capabilities of the malicious file or software.

## **2.7.6 Malware Analysis Environment Setup Overview**

The virtual environment will be set up according to the two malware analysis techniques viz. Static and Dynamic analysis.

### **2.7.6.1 Malware Static Analysis Environment**

A Static malware analysis environment provides the malware analyst with a glance of the actual behaviour and nature of the malware from the information gathered when the malware is at rest (Elisan, 2015). The purpose of this environment is to house the different malware analysis tools that will be used to conduct experiments for the purpose of this research.

The Static analysis environment does not require any dependencies as malware will not be executed in this environment; only the behaviour of the malware will be analysed.

According to Elisan, to have an effective and properly configured environment, the following characteristics must be considered when configuring the static analysis environment (Elisan, 2015):

- i. Can the malware analysis tools be installed and configured regardless of the type of operating system that are intended or designed for?
- ii. Can it offer possible mitigation techniques to possible system infection through hardening the environment in which the malware is being analysed?
- iii. Can it offer possible mitigation of the environment from being used as a jump box to perform further attacks?
- iv. Can the malware access online resources or services anonymously without any detection?

The above characteristics are only intended to serve as a guide when configuring the static analysis environment and to ensure that the malware that is being analysed does not cause any unintended infections.

**The setup of the static analysis environment is implemented with the following systems:**

- a) REMnux
- b) Incident Response Malware Analysis (IRMA)
- c) Windows 7 Virtual Machine

Figure 26, below is a graphical view of the toolsets found in the REMnux system.

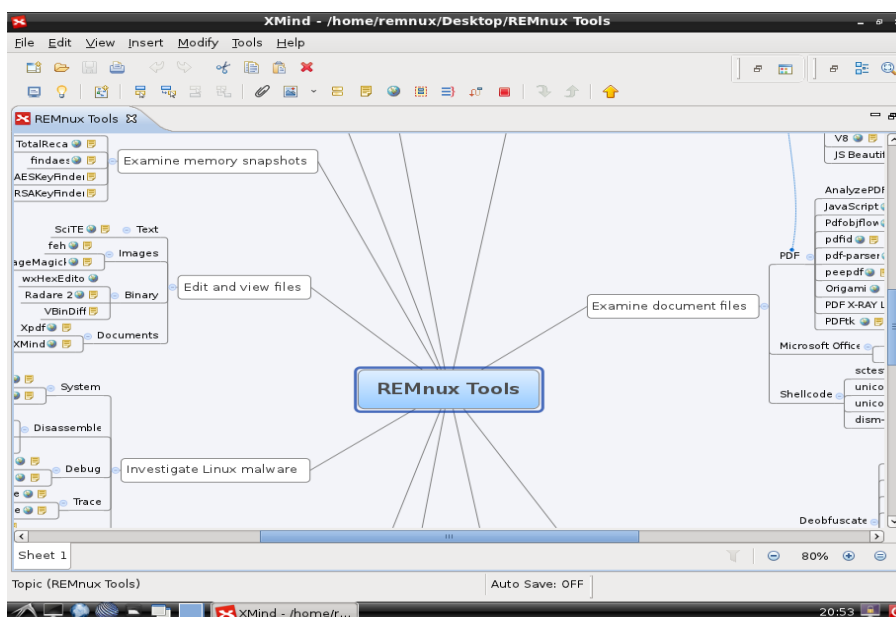


Figure 26: REMnux Architecture Overview (Zeltser, 2016)

Figure 27, provides a graphical overview of the architecture behind the IRMA system discussed in section 2.7.4.5 Incident Response Malware Analysis (IRMA).

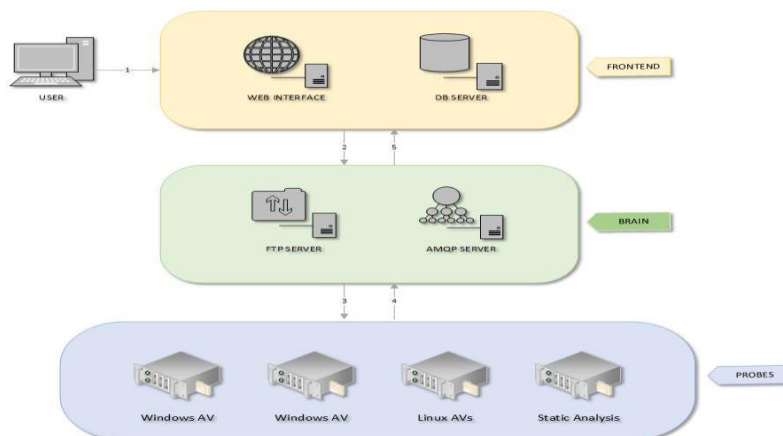


Figure 27: IRMA Architecture Overview (Quint et al., 2016)

### 2.7.6.2 Malware Dynamic Analysis Environment

Dynamic analysis environment can make or break an analysis session, especially if the malware is aware of this environment that is executed on (Elisan, 2015). Even though a lot of malware that are in the wild today do not possess evasion techniques, it is important that virtual environment evasion counter-measures are implemented.

According to Elisan this environment is an active computer and depending on the malware that is being executed, the environment may be configured to have access to the internet to facilitate malware communication behaviour.

It is then important that the environment is anonymized to protect the environment from being detected by the malware author or cyber criminals. The environment must also be isolated (Elisan, 2015). The sandbox systems that is set up include:

- a) Cuckoo Sandbox
- b) Windows 7 Virtual Machine
- c) Ubuntu 15.10 Virtual Machine

### 2.7.7 Summary of the Malware Analysis Environment Design Overview

Figure 28, provides a summary and the design overview of the malware analysis environment that was implemented and used as part of the research.

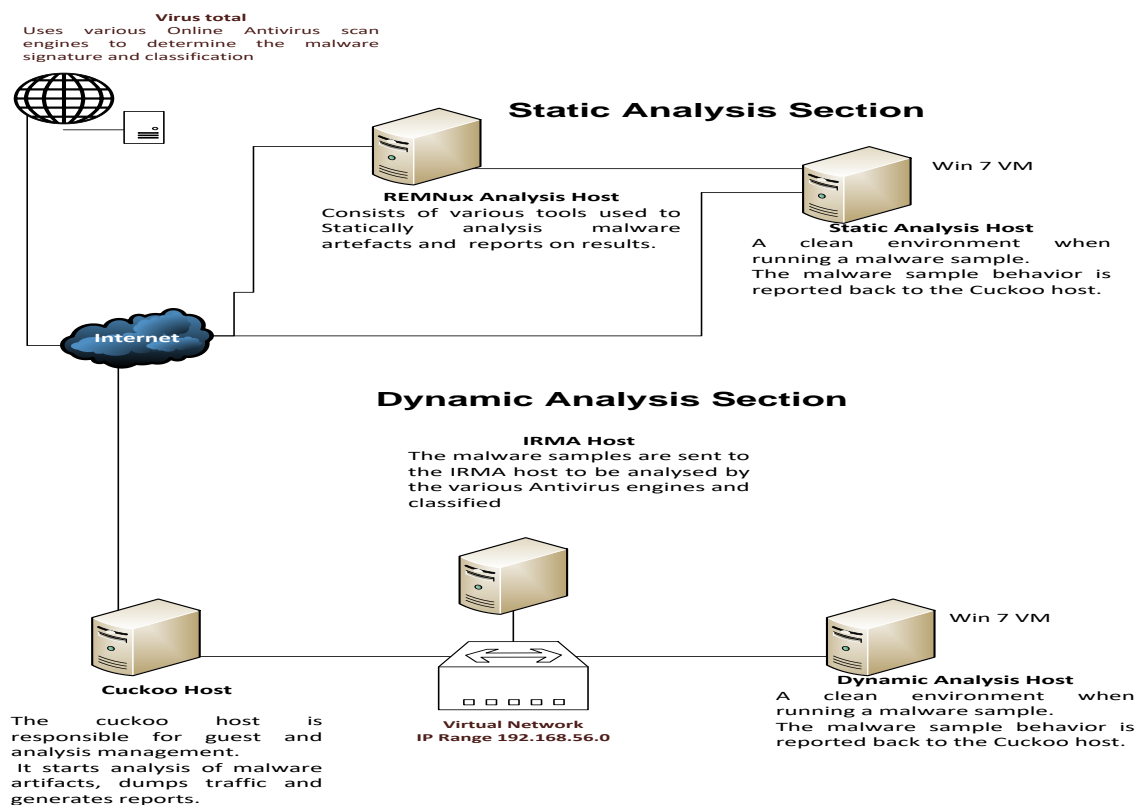


Figure 28: Malware Analysis Environment Overview

## 2.8 Chapter Summary

Foundational theory and background on malware and malware analysis, which are vital for this research and to successfully conduct experiments in chapter 4, were discussed. We deliberated on the history of malware, types of malwares such as trojans, botnets, viruses, etc., that fall under the category umbrella of malware. We also looked at various categories of malware and its characteristics. We looked at how each category of malware works and what sort of damage they cause through their different attacks. We found that all fall under the category of malware also known as malicious software.

Antivirus solutions or vendors are constantly challenged by the increased frequency of malware attacks. The malware analysis and detection approaches that are available to minimise the spread of malware or infected programs were discussed. The goals of malware analysis were outlined that emphasised the importance of the malware analysis process.

The chapter dives deeper into the two types of malware analysis viz. Static and Dynamic analysis. The two techniques were described in detail and it was established that Static analysis is a manual process that required analysing the properties of malware sample, while Dynamic analysis required executing the malware sample in an isolated environment in order to monitor and record the behaviour of malware. The limitations of these two techniques were also outlined.

The new techniques used by Malware authors deploying methods such as code obfuscation or altering the behaviour of malware in order to create zero-day malware that can evade controls detection mechanisms, were also discussed. We provide a selective reference to some of the literature on malware analysis and also a clear understanding of the existing malware detection techniques.

Malware analysis and detection techniques presented in this literature review chapter were based on several strategies of analysis that are commonly used by malware analysts. We further explained the functionality of malware by focusing on areas such as the behaviour of malware and covert malware launching techniques. This allowed diving deeply into the behaviour of malware, covert malware launching techniques and malware anti-reverse-engineering techniques that focused on code obfuscation, packers, VBA macros and portable executables.

Finally, the architecture, setup of the malware analysis lab and specific Static and Dynamic analysis tools, which allowed the malware experiments in chapter 4 to be conducted, were also described in detail.

# CHAPTER 3

## RESEARCH METHODOLOGY

### CHAPTER OVERVIEW

This chapter focuses on the following areas:

- Introduction
- Research Design
- A Model of the Research Process
- Research Methodology
- Data Preparation and Data Sources
- Data Collection Techniques
- Sampling Techniques
- Quantitative Data Analysis of Collected malware
- Research Methodology Actions Overview
- Chapter Summary

This chapter discusses the research methodology used as part of this dissertation. This chapter seeks to explain in detail the methodology followed throughout the research in order to detect and analyse malware in an isolated environment.

### 3 INTRODUCTION

The focus of this chapter is to explain the methods that are used to conduct and analyze the research. This chapter gives information about the positivism paradigm, research process, sampling strategy, data collection and source, sampling techniques, and data analysis methodology.

#### 3.1 Research Design

Research design can be described as a general plan about what you will do to answer the research question (Dudovskiy, 211; Saunders. et al., 2012). According to Bryman and Bell (Bryman and Bell, 2007), there are five types of research designs viz.:

- **Experimental design** – This design is unusual in management research due to the challenges of accomplishing exact control levels when dealing with organizational behaviour



- **Cross-sectional design or social survey design** - This design involves collecting data on several cases during the same time frame in order to gather qualitative or quantitative information that is related to two or more variables, in an effort to determine associations between the variables after the data has been analyzed
- **Longitudinal design** - This design is specifically used to monitor changes over time in the applicable research environment
- **Case study design** - This design is an intensive examination of a particular situation or instance
- **Comparative design** – This design is where identical or contrasting cases are studied, and the similarities or differences are reported

Research design can be divided into two groups: exploratory and conclusive.

- **Exploratory research design** - according to its name it merely aims to explore specific aspects of the research area and does not aim to provide ultimate and conclusive answers to the research questions proposed. In exploratory research the researcher may even change the direction of the study to a certain extent, however not fundamentally, according to new evidence gained during the research process (Saunders. et al., 2012) as cited by (Dudovskiy, 2011).
- **Conclusive research design** - as the name implies, this research design applies to create findings that are practically useful in decision-making and also reaching conclusions. Conclusive research design typically involves the application of quantitative methods of data collection and data analysis. Furthermore, conclusive studies tend to be deductive in nature, and objectives of research in these types of studies are accomplished through testing the hypotheses. Conclusive research is also divided into two categories which are:
  - causal
  - descriptive

Descriptive research design describes the following in the research area (Saunders. et al., 2012) as cited by (Dudovskiy, 2011):

- specific elements
- causes
- or phenomena

Table 3 illustrates the main differences between exploratory and conclusive research in relation to important components of the dissertation (Dudovskiy, 211; Saunders. et al., 2012).

Table 7: Differences between exploratory and conclusive research (Source: Pride & Ferrell, (2007)

Research project components	Exploratory research	Conclusive research
Research purpose	<b>General:</b> to generate insights about a situation	<b>Specific:</b> to verify insights and aid in selecting a course of action
Data needs	Vague	Clear
Data sources	Ill defined	Well defined
Data collection form	Open-ended, rough	Usually structured
Sample	Relatively small; subjectively selected to maximize generalization of insights	Relatively large; objectively selected to permit generalization of findings
Data collection	Flexible; no set procedure	Rigid; well-laid-out procedure
Data analysis	Informal; typically non-quantitative	Formal; typically quantitative
Inferences/recommendations	More tentative than final	More final than tentative

Our research uses the conclusive research approach that is described in detail in the following sections.

### 3.2 A Model of the Research Process

A probable model of the research process that is discussed and represented in diagrammatic form by Oates (Oates, 2007) that is presented in figure 29 below.

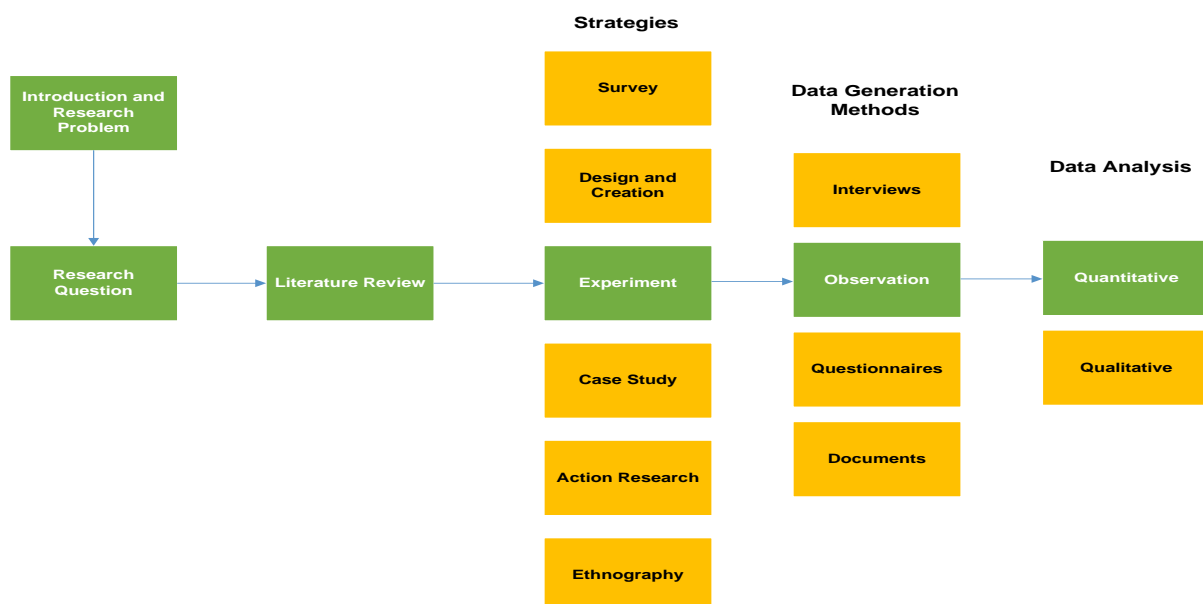


Figure 29: Model of the research process showing the variety of paths that can be undertaken (Brand, 2010)

The process outlined in figure 29 above assists in following a way to navigate from the introduction and problem statement to formulating the research questions and to determining the answers to the research questions. The particular model presented by Oates shows that the introduction and research problem statement are inputs into developing proper and meaningful research questions. The objective of this starting phase of the process is to show why this line of research is essential.

It shows how it has been fully addressed in the literature review and how the research will be used. The research questions are the fundamental thread throughout the entire process of this research. After they have been formulated, they are then used to select an appropriate research strategy, data generation method and data analysis method. The research questions that are formulated in this thesis are clearly visible throughout the research process, and arriving at answers to an enquiry is dependent upon the selection of the suitable research strategy, data generation method and data analysis methods suitable for the questions, being in chapter 1.

Major consideration is required as to the choice of the research model before the selection process of research method starts. There are various research paradigms that exist to guide us through this research (Oates, 2007, Guba and Lincoln, 1994) these include the following:

- positivism
- interpretivism
- critical research

Methodology is described as a framework associated with a particular set of paradigmatic assumptions that is used to conduct research (O'Leary, 2004). According to Guba and Lincoln (Guba and Lincoln, 1994), the "Questions of method are secondary to the questions of paradigm that defines the basic belief system or broad view that guides the investigator or researcher, not only in choices of method but in ontologically and epistemologically introductory approaches." The aforementioned statement is significant in the sense that it stresses the order in which research can be conducted. A research paradigm is referred to as an appropriate and overarching, philosophical viewpoint that must be adopted by the researcher (Brand, 2010).

### **3.2.1 Research Paradigms**

Research paradigms include positivism, interpretivism and critical research as previously stated. According to Oates (Oates, 2007), a paradigm is explained as "a set of shared assumptions or ways of thinking about some aspect of the world".

### **3.2.1.1 Positivism**

Positivism adheres to the view that only genuine knowledge is gained via observing that includes measurement and trustworthiness. It often involves the use of existing theory to develop hypotheses to be tested during the research process (Dudovskiy, 2011). Positivists have faith in the authority of the knowledge that is formed by empirical and verifiable proof (Burrell and Morgan, 1979). Positivism relies on the following aspects of the science (Dudovskiy, 2011):

- Science is deterministic
- Science is mechanistic
- Science uses method
- Science deals with empiricism

According to Oates (Oates, 2007), positivism is the basis of the experimental method that also consists of two basic assumptions:

- The world has order, is regular and is non-random
- The world can be investigated objectively

These assumptions are important because they facilitate the discovery of consistencies, forms and laws through conducting experimentation to discover evidence of cause and/or effect. The discovery process is initiated by the formulation of a hypothesis that is followed by experiments designed to refute or confirm the hypothesis (Brand, 2010).

Sureness in a theory may be increased each time it fails to be disproven. Controlled experiments are typically used by positivist researchers but they are not limited to the use of controlled experiments as their research strategy. Surveys are other strategies that are regularly used by this paradigm. The positivists' paradigms are considered to be reductionist as well. For example, they study phenomena by breaking them down into less complicated components.

According to Guba and Lincoln (1994) and Brand (Guba and Lincoln, 1994, Brand, 2010) the positivism methodology is described to be experimental and manipulative. "Questions and/or hypotheses are stated in propositional form and subjected to empirical tests to verify them; possible confounding conditions must be carefully controlled (manipulated) to prevent outcomes from being properly influenced".

### **3.2.1.2 Interpretivism**

Interpretivist approach is based on naturalistic approach of data collection such as interviews and observations (Dudovskiy, 2011). In comparison to positivism, interpretivism does not pursue to verify or refute a hypothesis. The interpretivist tries to understand the phenomena through the meanings and values assigned to them by people. This means that multiple,

subject certainties are exhaustive, and hence there is no single truth or fact. Different researchers can view the world differently and their values and actions mold the research process (Brand, 2010). This can result in multiple interpretations, and data collected through this paradigm is usually qualitative (Guba and Lincoln, 1994).

According to Walsham (Walsham, 2006), “Interpretive methods of research start from the position that our knowledge of reality, including the domain of human activities which is a social building block by human actors and that this applies equally to scholars or researchers”

Klein and Myers suggested the use of interpretivism approach in business studies that involves the following principles:

- The Fundamental Principle of the Hermeneutic Circle
- The Principle of Contextualization
- The Principle of Interaction between the Researchers and the Modules
- The Principle of Dialogical Reasoning
- The Principle of Multiple Interpretations
- The Principle of Generalization and Abstraction
- The Principle of Suspicion

Table 8 provides a summary of the basic differences between positivism and interpretivism are illustrated by Pizam and Mansfeld as cited by Dudovskiy (Dudovskiy, 2011).

Table 8: Positivism and Interpretivism (Bajpai, 2011)

<b>Assumptions</b>	<b>Positivism</b>	<b>Interpretivism</b>
Nature of reality	Objective, tangible, single	Socially constructed, multiple
Goal of research	Explanation, strong prediction	Understanding, weak prediction
Focus of interest	What is general, average and representative	What is specific, unique, and deviant
Knowledge generated	Laws Absolute	Meanings Relative
Subject / Researcher relationship	Rigid separation	Interactive, cooperative, participative

Assumptions	Positivism	Interpretivism
Desired information	How many people think and do a specific thing, or have a specific problem.	What some people think and do, what kind of problems they are confronted with, and how they deal with them.
Research approach	Deductive	Inductive
Ontology	Objective	Subjective
Axiology	Value-free	Biased
Research strategy	Quantitative	Qualitative

### 3.2.1.3 Critical research

Critical research is comparable to interpretivism from the standpoint that there are multiple views of reality, but differs in a sense that it says social certainty owns objective properties that interpretivists discount (Brand, 2010). According to Oates and Brand (Brand, 2010, Oates, 2007)) “Critical researchers seek to identify and challenge the conditions of domination, and the restrictions and unfairness of the status quo and taken-for-granted assumptions”.

Transactional and subjectivist are the epistemology and historical realism as the ontology of critical research (Guba and Lincoln, 1994).

### 3.2.1.4 Research Paradigm Selected for this Research

This research has adopted the positivist paradigm as the selected approach that seeks to address the research questions of this thesis. An empirical approach is appropriate for this research because the result should be similar or the same, regardless of how it is measured. The results produced should be similar or the same by the various tools that are used to perform measurement.

Empirical approach relies on objective facts that have been established and can be demonstrated (Dudovskiy, 2011). This approach is undertaken in order to provide a practical basis that ensures that a reasonably objective measurement of the purpose of the study is provided. Empirical research also involves the collecting and interpreting evidence through methods such as surveys, interviews, experimentation and observation. The empirical approach follows the quantitative research approach that will be discussed later in the chapter.

(Perry et al., 2000) listed the steps to conduct an empirical study:

- formulation of a hypothesis or question to test

- seeing a situation
- abstracting observations into information
- analyzing the information and
- drawing of conclusions with respect to the tested hypothesis

There are five types of empirical research in which data can be created that include that which are listed by (S.Easterbrook et al., 2008):

- Controlled Experiments
  - Case Studies
  - Survey Research
  - Ethnographies
  - Action Research
- Controlled Experiments** - One or more independent variables are manipulated to measure the effect of one or more dependent variables that will assist the researcher to be able to determine how the variables are linked and to classify causality
  - Case Studies** - Investigates an occurrence within a context and reveals causality. They are used where the reductionism of a controlled experiment is inappropriate. This could include when effects may take a long time to appear or where the context plays a role in the phenomena under investigation (S.Easterbrook et al., 2008)
  - Survey Research** – They are conducted through the use of questionnaires, structured interviews, and data logging to pinpoint characteristics of a representative sample from defined populations
  - Action Research** – It focuses on the resolutions of real world problems
  - Ethnographies** – It is defined by (S.Easterbrook et al., 2008) as “Ethnography is a form of research focusing on the sociology of meaning through field observation”

Selecting the most appropriate research method requires considering ontology, epistemology, methodology, resources and the abilities of the researcher with respect to the phenomena under investigation (S.Easterbrook et al., 2008, Brand, 2010). Empirically based questions can be asked to facilitate comprehension of the ontology of the phenomenon. The first steps as stated by (S.Easterbrook et al., 2008), recommend selecting the research strategy that clarifies the research question(s). This starts by examining exploratory questions to aid in understanding the phenomena. Such questions assist in the resolution of measurable and valid evidence. Research questions in this dissertation, as specified in chapter 1, are exploratory in nature and can be answered in a literature review and through empirical approach by means of experiments.

### 3.2.2 Selected Empirical Research Method

All of the research approaches already discussed above would be suitable for addressing the research questions in chapter 1 of this thesis. For the purpose of our research, these research questions are basically exploratory in nature and the empirical research approach selected for this research is conducted through controlled experiments.

### 3.3 Research Methodology

The business research methods are defined as “a systematic and scientific procedure of data collection, compilation, analysis, interpretation, and implication pertaining to any business problem” (Bajpai, 2011). According to Dudovskiy (Dudovskiy, 2011) the types of research methods are classified into several categories according to the nature and purpose of the study and other attributes. The types of research methods are divided into two categories namely:

- Quantitative
  - Qualitative
- a) **Quantitative research**, “describes, infers, and resolves problems using numbers. Emphasis is placed on the collection of numerical data, the summary of those data and the drawing of inferences from the data” (Herbst and Coldwell, 2004) as cited by (Dudovskiy, 2011). This type of research methodology is also classified as analytically based on the nature of the research. Analytical research, on the other hand, is fundamentally different in a way that “the researcher has to use facts or information already available and analyze these in order to make a critical evaluation of the material” (Kumar, 2008)
  - b) **Qualitative research**; this method is based on emotions, feelings, sounds and other non-numerical and unquantifiable elements. It has been noted that “information is considered qualitative in nature if it cannot be analyzed by means of mathematical techniques. This characteristic may also mean that an incident does not take place often enough to allow reliable data to be collected” (Herbst and Coldwell, 2004) as cited by (Dudovskiy, 2011). This type of research methodology is also classified as descriptive based on the nature of the research. Descriptive research usually involves surveys and studies that target to detect the facts. This means that descriptive research mostly deals with the “description of the state of affairs as it is at present” (Kumar, 2008)

#### 3.3.1 Selective Research Method - Quantitative Research

For the purpose of this dissertation we have selected the quantitative research approach. It is based on measuring quantitative malware samples and it is empirical because it is a data-



driven research and any conclusions in the research can be verified by experiments and observations. The purpose of the research is explorative because we plan to explore the available empirical malware data to see whether it is possible to gain any insight into the behaviour of the malware samples that are being analyzed (Tajalizadehkhooob, 2013).

Figure 29 provides an overview of the actions that are going to be taken in this research and each individual step will be explained in detail in the upcoming sections. This study is also aligned with the positivist research paradigm.

### 3.4 Data Preparation and Data Source

Live malware data samples were collected from an email filtering, in cases whereby sensitive information is found from malware samples, permission was then requested from the company making use of the email filtering system. Online data sources or repositories were also utilized to outsource malware samples or artifacts for further analysis.

### 3.5 Data Collection Techniques

Data collection methods allow us to scientifically collect information about our entities of study (phenomena, objects people) and about the settings in which they occur (Chaleunvong, 2009, Morgan and Harmo, 2001).

There are various types of data collection techniques used with human participants when conducting research. According to Morgan and Harmon (Morgan and Harmo, 2001) “research approaches or designs are approximately orthogonal to the techniques of data collection, and thus, in theory, any type of data collection technique could be used with any approach to research”. The following are the various types of data collection techniques:

- Using available information
- Observing
- Interviewing (face-to-face)
- Administering written questionnaires
- Focus group discussions

Table 9 provides a summary of data collection techniques used by researchers.

**Notes for table 2 below:** Symbols indicate likelihood of use (++ = quite likely; + = possibly; – = not likely).

Table 9: Data Collection Techniques Used by Research Approaches (Morgan and Harmo, 2001)

Data Collection Techniques	Research Approach		
	Quantitative Research		Qualitative Research
	Experimental & Quasi-Experimental	Comparative, Associational, & Descriptive Approaches	
<b>Research Report Measure</b>			
Physiological recordings	++	+	-
Coded observations	++	++	+
Narrative observations	-	+	++
Participant observations	-	+	++
<b>Other Measures</b>			
Standardized Tests	+	++	-
Archival measures/documents	-	+	++
Content analysis	-	+	++
<b>Self-report measures</b>			
Summated attitude scales	+	++	-
Standardized personality scales	+	++	-
Questionnaires (surveys)	+	++	+
Interviews	+	++	++
Focus groups	-	-	++

For the purpose of our research we have selected the observation technique. Observation is a technique that involves scientifically selecting, observing and recording behaviour and characteristics of living creatures, phenomena or objects. In our research we will be performing experiments, observe and record the behaviour of experiments conducted. Using this approach, the researcher observes and records the behaviours of the experiments rather than relying on reports from other researchers (Morgan and Harmo, 2001).

The primary tool for the collection of data will be the live malware from online malware repository or the company's email filtering system. The collected samples will then be analyzed in a sandbox environment, the results of which are discussed in chapter 5.

### 3.6 Sampling Techniques

A proper Sampling strategy is a crucial part of any type of research. Sampling is “the process or technique of selecting a suitable sample from the whole population” in order to determine and generalize characteristics or parameters (Adams et al., 2007). Sampling techniques can be divided into two categories: probability and non-probability.

According to Fox and Bayat (Fox and Bayat, 2007) probability sampling is used when every element of the population has a known and has no zero chance of being included in the sample.

Bryman and Bell (2007) explain non-probability sampling as an “*umbrella term for a wide range of the types of sampling strategy based on common sense and best personal judgment*”, that are outside the probability sampling category.

Table 10: Sampling Techniques (Dudovskiy, 2011)

Probability Sampling	Non-probability Sampling
In probability sampling, each population member has a known, non-zero chance of participating in the study. Randomization or chance is the core of probability sampling techniques.	In non-probability sampling on the other hand, sample group members are selected non-randomly, therefore, in non-probability sampling only certain members of the population has a chance to participate in the study.

According to Dudovskiy (Dudovskiy, 2011) probability sampling consists of the following sampling techniques :

- Simple random sampling
- Stratified random sampling
- Systematic random sampling
- Multistage random sampling
- Cluster sampling

Probability Sampling also consists of two advantages as listed in table 11 below.

Table 11: Advantages and Disadvantages of Probability Sampling (Dudovskiy, 2011)

Advantages	Disadvantages
The absence of systematic and sampling bias	Higher complexity compared to non-probability sampling
Higher level of reliability of research findings	More time consuming
Increased accuracy of sampling error estimation	Usually more expensive than non-probability sampling
The possibility to make inferences about the population	None

For the purpose of our research, we have selected the simple random sampling technique that is part of the probability sampling category. The samples were randomly selected from the email file attachment that was suspected to contain malware by the system.

Simple random sampling (also referred to as random sampling) is the purest and the most straightforward probability sampling strategy. In the simple random sampling technique, each member of the population is equally likely to be chosen as part of the sample. It has been stated that “the logic behind simple random sampling is that it removes bias from the selection procedure and should result in representative samples” (F.J.Gravetter and L.B.Forzano, 2011, Dudovskiy, 2011).

We randomly selected malware samples among those submitted from the online email filtering systems and only downloaded emails labelled to have been suspected of having malware attachments. To ensure our datasets were as diverse as possible, we selected samples having different file formats and not belonging to different malware families.

### 3.7 Quantitative Data Analysis of Collected Malware

Dudovskiy (Dudovskiy, 2011) stated that in quantitative data analysis we are expected to turn raw numbers into meaningful data through the application of rational and critical thinking. The same figure within a data set can be interpreted in different ways; therefore it is important to apply fair and careful judgement.

This section is used to support the examination of the research questions as described in chapter 1, primarily with respect to the ability to analyze malware in order to determine detection on enterprise systems. Results from the practical experiments that were conducted from the live malware data sets samples were collected. The results that were obtained were

then interpreted to differentiate between malicious and kind of samples with minimal false positives and false negatives and also taking into consideration the problem statement, questions, and the objectives of the proposed study. The results of the experiments conducted are presented in chapter 6 of this dissertation.

### 3.8 Research Methodology Actions Overview

Figure 30, provides an overview summary of the research methodology used as part of this research study.

- The Malware analyst described in the diagram had to search for malware samples that were experimented on and analyzed as part of this research.
- These malware samples were downloaded from well-known malware database repositories.
- The malware samples that were downloaded were then stored in environment implement to prevent the malware from spreading to other environments.
- The samples were then experimented on using the two types of malware analysis techniques that are discussed in literature.
- Once all the experiments were concluded results were finalized and analyzed using quantitative methods.
- The results were then documented and discussed in detail in chapter 5.

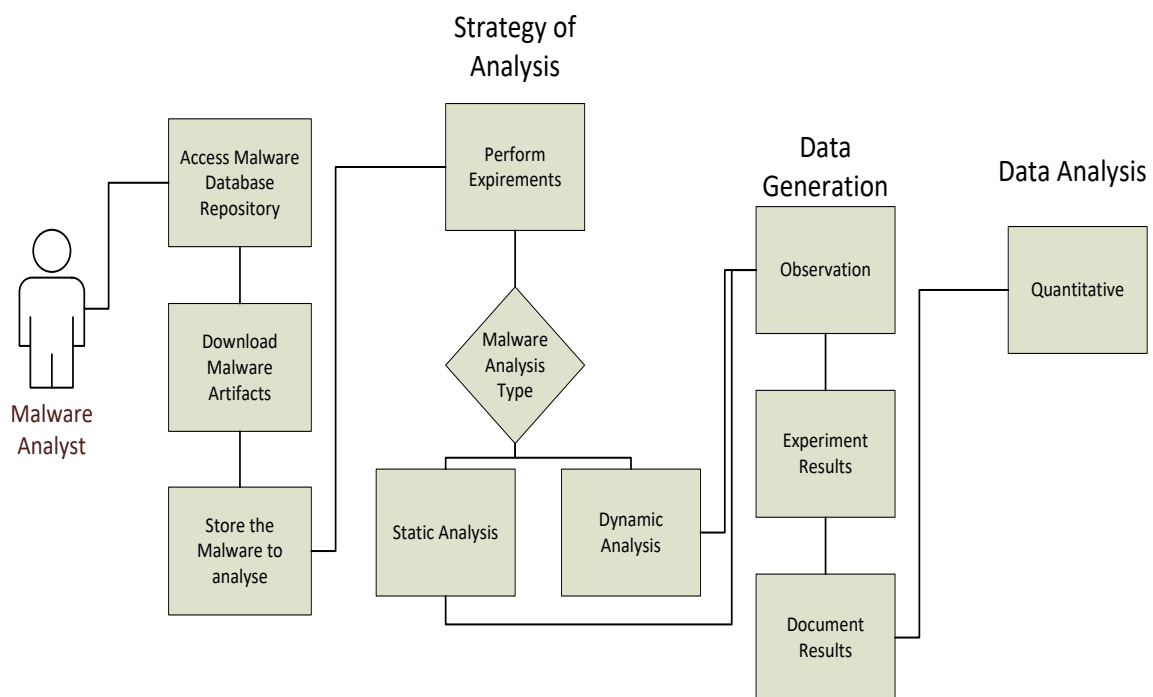


Figure 30: Research Methodology Actions Overview

### **3.9 Chapter Summary**

This chapter covered the research design, model of the research process and research methodology. It also provided a brief overview on the observation technique used in the collection and analysis of data. This research adopted the positivist paradigm as the selected approach that seeks to address the research questions of this dissertation. The sampling approach was also explained. For this dissertation, we have selected the simple random sampling technique that is part of the probability sampling category. The quantitative research method was discussed and chosen as an appropriate research methodology for this dissertation. This methodology is based on measuring quantitative malware samples and it is empirical because it is a data-driven research and any conclusions in the research can be verified by experiments and observations.

It was mentioned that the purpose of the research is explorative because it planned to explore the available empirical malware data to see whether it is possible to gain any insight into the behaviour of the malware samples that would be analyzed. Finally, the quantitative data analysis procedure used in this thesis was briefly discussed. A high level diagram (Figure 30) provided an overview of the research process of this research. Chapters 4 and 5 provide quantitative analysis and interpretation of the empirical findings in line with the objectives of this dissertation.

# CHAPTER 4

## MALWARE ANALYSIS EXPERIMENTS

### CHAPTER OVERVIEW

This chapter focuses on the following areas:

- Introduction
  - Malware Analysis Process Overview
  - Malware Analysis Experiments
- Chapter Summary

### 4 INTRODUCTION

Malware analysis experiments are conducted as described in chapter 2; the literature review and preliminary experiment results were then briefly discussed in this chapter. This chapter kicks off with an overview of the malware samples or artifacts; it also discusses the analysis process before detailing the experiments conducted and their results.

#### 4.1 Malware Analysis Experiments

In this section the results from malware analysis experiments conducted on two malware artifacts of which one was a Microsoft word format and the other a portable executable, were discussed. The following malware artifacts were analysed and investigated:

**a) Microsoft Word Files:**

- Ticket\_354041

**b) Portable Executables:**

- 1123211-090SD.exe

Experiments on additional analysis and results of other malware artifacts are available on the provided CD as indicated in Appendix F. Analysis on the above mentioned artifacts are carried out as described in the Literature Review in chapter 3 of this research. The experiments started with static analysis and finished with dynamic analysis of the aforementioned malware artifacts.

##### 4.1.1 Malware Analysis Process Overview

The malware analysis process follows a simplified method of first analysing the static or code properties of the malware artifact; this is done in order to have an understanding of the

characteristics of the artifact. The artifact is analysed using various static or code analysis tools that have been installed in our isolated environment.

The process then follows an automated approach where the artifact is executed in the isolated environment and uploaded to the online services to find the classification of the malware. The process flowchart in figure 31 below provides an overview of the analysis process that was followed in this chapter to perform malware experiments.

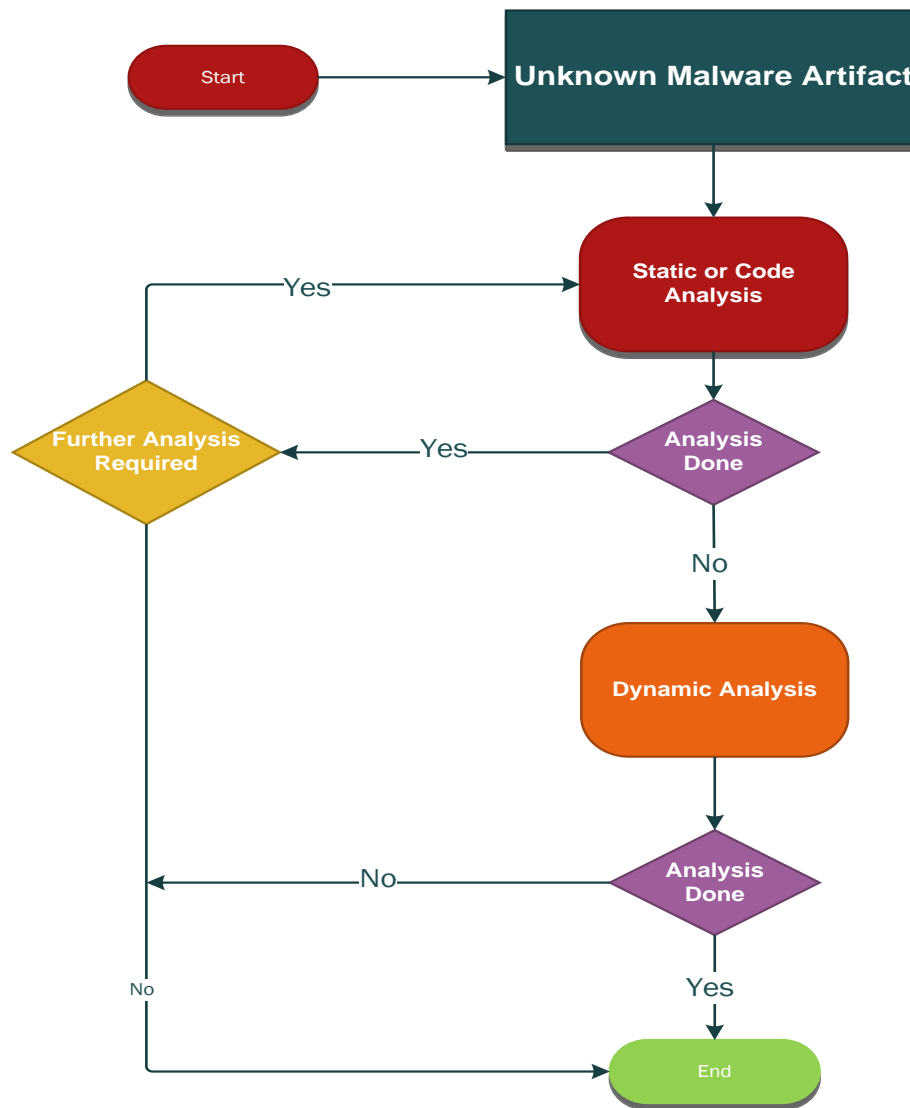


Figure 31: Malware Analysis Process Flowchart

According to the Malware analysis template proposed by Zeltser (Zeltser, 2015), the analysis process focuses on specific areas of the analysed artifact in order to determine if the artifact is malicious or not. The following are the focus areas of analysis of the artifact that are split into two areas indicating the tools used for each area; this is not an exhaustive list of tools used:



#### a) Static Analysis Focus Areas

- PE Section or File Hashes (rhash, PEstudio, CFF Explorer)
- File Properties (File command, PEstudio, CFF Explorer): Description, version, file header characteristics
- File Signatures
- Indicators of compromise (IOC, pescanner, PEstudio, pedump, peframe, OfficeMalScanner)
- Strings (strings, CFF Explorer, pestr): Functions, domains, IP addresses, commands, error messages, Strings, CALLs, program flow, loops
- Packed (pescanner, PEiD, PEstudio, peframe)
- Entropy (pescanner, CFF Explorer): File, sections
- Imported/Exported Functions (PEStudio, CFF Explorer / Dependency Walker)
- Compile Time (pescanner)
- Open Source Research (VirusTotal)

#### a) Dynamic Analysis Focus Areas

- File System Artifacts Properties (Cuckoo)
- Indicators of Compromise
- Triggers: Browser, mail client, specific web pages (Google, bank), time, reboot, user/admin privileges
- Signatures
- Dependencies: DNS, HTTP, IRC, ARP
- Network Artifacts (INetSim,): C2 domains/IP addresses, protocols, user-agent
- Memory Analysis (Volatility): rogue processes, code injection, rootkits, network artifacts
- Open Source Research (VirusTotal)

## 4.2 Malware Analysis Experiments

This section describes the experiments conducted on the selected malware samples using both Static and Dynamic analysis techniques of Malware analysis.

### 4.2.1 Analysis of Malware artifact - Ticket\_354041

#### 4.2.1.1 Static Analysis

The malware artifact was analysed in a REMnux sandbox and Windows 7 environments as described in detail in chapter 5. Before the analysis of the malware artifact acquired could

commence it was important to fingerprint the file, confirm the type of file and properties of the artifact in order to determine the type of file that was analysed.

### a) File Section Hashes

The first step was to uniquely identify the malware artifact in order to be able to fingerprint or identify the malicious artifact in future. MD5, SHA1 and SHA256 sums of the artifact were created and these hashes themselves were enough to be able to fingerprint the file. This part of the process was achieved by running the artifact through the “*rhash*” tool that is a console utility for computing and verifying hash sums of files. By executing the *rhash* command with a combination of *-M* (for MD5) or *-H* (SHA1) or *-sha256* hash values of file were provided.

#### **rhash command execution:**

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$ rhash -M
ticket_354041.doc
543c0cf636bc0e56007e6211cd05ecf2 ticket_354041.doc
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$ rhash -H
ticket_354041.doc
400cb9f479fd5ab09aa895245e16ba999ce5142e ticket_354041.doc
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$ rhash --
sha256 ticket_354041.doc
3ea894203c48d37b73ce9202dec7eedbf1c724b707f7de058e42c18c3e55bd49
ticket_354041.doc
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
```

Figure 32: Listing of File Section Hashes – ticket\_354041.doc

The output results above indicate that the artifact’s MD5, SHA1 and SHA256 hashes are as follows:

MD5 = 543c0cf636bc0e56007e6211cd05ecf2

SHA1 = 400cb9f479fd5ab09aa895245e16ba999ce5142e

SHA256 = 3ea894203c48d37b73ce9202dec7eedbf1c724b707f7de058e42c18c3e55bd49

These hashes were later confirmed from the other tools that were used as part of performing static analysis.

### b) File Properties

The second setup was to then confirm the properties and type or format of the file that we were using. The process started by first executing the “*file*” command against the 'ticket\_354041.doc' artifact. The results of the command are as follows:

### File command execution:

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$ file
'ticket_354041.doc'
ticket_354041.doc: Composite Document File V2 Document, Little Endian,
Os: Windows, Version 6.1,
Code page: 1251,
Title: ,
Author: Laura,
Template: Normal.dot,
Last Saved By: Windows, Revision Number: 1,
Name of Creating Application: Microsoft Office Word,
Total Editing Time: 01:00,
Create Time/Date: Wed Oct 19 15:33:00 2016,
Last Saved Time/Date: Wed Oct 19 15:34:00 2016,
Number of Pages: 1, Number of Words: 0,
Number of Characters: 2, Security: 0
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
```

Figure 33: Listing of File Properties - ticket\_354041.doc

The output results above indicate that the file is a Composite Document file written using the Microsoft Office word to execute on Windows operating system. The file consists of one page and it was created on 19 October 2016 by Laura.

### c) File Signature

The third step of the analysis process was to determine the malware signatures of the file by running the “Clamscan” tool against the file.

### Clamscan tool execution:

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$ clamscan ticket_354041.doc
LibClamAV Warning: *****
LibClamAV Warning: *** The virus database is older than 7 days! ***
LibClamAV Warning: *** Please update it as soon as possible. ***
LibClamAV Warning: *****
ticket_354041.doc: OK

----- SCAN SUMMARY -----
Known viruses: 3832461
Engine version: 0.98.7
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 0.29 MB
Data read: 0.15 MB (ratio 1.92:1)
Time: 5.911 sec (0 m 5 s)
```

Figure 34: Listing of File Signatures - ticket\_354041.doc

The results from the Clamscan output indicate that the file was not infected but again this could have been a false negative, and further analysis was required.

#### d) Indicators of Compromise (IOC)

The fourth step of analysis was to determine the IOCs using the PEstudio application.

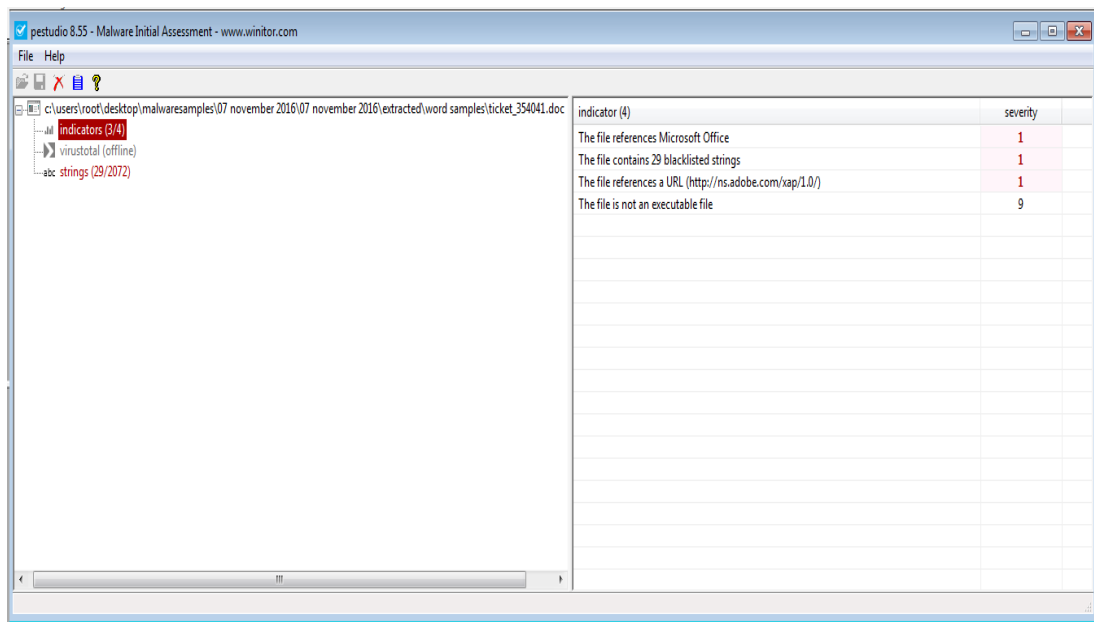


Figure 35: Indicators of Compromise - ticket\_354041.doc

Figure 35 above indicated that there were 3 out of 4 IOCs found from the artifact that was analyzed. These IOCs included the following:

- The files that were referencing Microsoft Office
- The file contained 29 blacklisted strings
- The file referenced a URL (<http://ns.adobe.com>)

#### e) Strings

Figure 36 below indicated that 29 out of 2072 malicious strings were found within the artifact. Some of the ASCII strings yielded interesting results that indicate application such as Microsoft Office application is executed and several other system processes are created (ASCII, size 10 - HeapCreate) or called (ASCII, size 15 – GetModuleHandle)

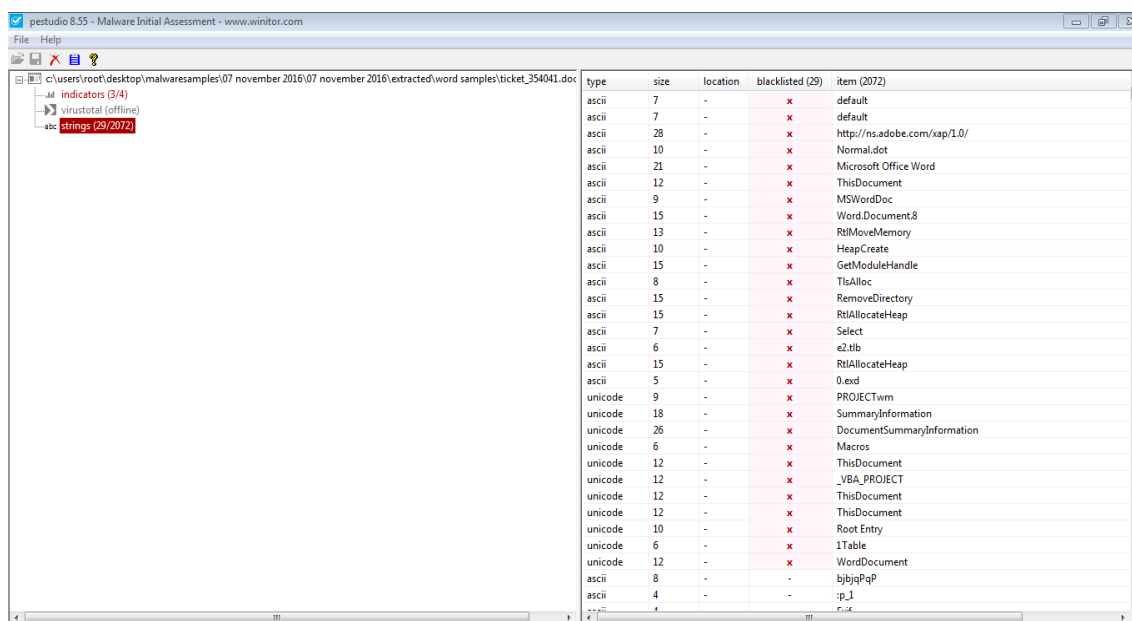


Figure 36: Blacklisted Strings - ticket\_354041.doc

#### f) Obfuscated Code

The first step was to then determine if the file contains Obfuscated Code with “balbuzard.py” tool as indicated from the results already discussed that the artifact might contain obfuscated code.

#### Balbuzard.py tool execution.

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
balbuzard.py ticket_354041.doc
```

Figure 37: Listing of Obfuscated Code - ticket\_354041.doc

The output results from the “balbuzard.py” tool provided over a 1000 line of information and detailed information, are available in appendix C. The results provided us with more information about which IOC sections are discussed below. Further analyses of the output results were broken down into seven sections and discussed in the following section.

### 1. Embedded URLs Found

The URLs below were found within the file; this indicates that the malware would try to access the URLs below in order to download additional files for the malware to be able to execute further.

```
URL (http/https/ftp) - 'http://ns.adobe.com/xap/1.0/x00'  
URL (http/https/ftp) - 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'  
URL (http/https/ftp) - 'http://ns.adobe.com/camera-raw-settings/1.0/'  
URL (http/https/ftp) - 'http://ns.adobe.com/photoshop/1.0/'  
URL (http/https/ftp) - 'http://ns.adobe.com/xap/1.0/'  
URL (http/https/ftp) - 'http://purl.org/dc/elements/1.1/'  
URL (http/https/ftp) - 'http://ns.adobe.com/xap/1.0/mm/'  
URL (http/https/ftp) - 'http://ns.adobe.com/xap.../sType/ResourceEvent#'
```

Figure 38: Listing of Embedded URLs - ticket\_354041.doc

## 2. Email Address

The below email address was found within the code of the file that indicates that the malware will try to send additional information about the victim using the email client on the infected system and will send an email to the email address below.

```
e-mail address - 'MLRdfP-1V2gSSWSDd-jQR--..Vdf0E2rLEGC6AD@NRCD.AC'
```

Figure 39: Listing of Embedded Email Address - ticket\_354041.doc

## 3. Packed Packages – Portable Executables

This section of the malware indicates that the malware is packed with a few portable executables that will execute other applications if it is executed on the targeted host.

```
at 0000673A: EXE MZ headers - 'MZ'  
at 0000C847: EXE MZ headers - 'ZM'  
at 0000CB95: EXE MZ headers - 'ZM'  
at 00005FAE: EXE PE headers - 'PE'  
at 00009C13: EXE PE headers - 'PE'  
at 00009C7D: EXE PE headers - 'PE'  
at 0002741E: EXE PE headers - 'PE'  
at 0002748E: EXE PE headers - 'PE'  
at 00027624: EXE PE headers - 'PE'  
at 000278A6: EXE PE headers - 'PE'
```

Figure 40: Listing of PE Headers - ticket\_354041.doc

## 4. Executable File Commands

This section indicates some executable commands or applications mentioned on Packed Packages section above; the malware looks for the Adobe application and if the application is present it will then execute Adobe by using the below executable commands:

```
at 00003A8E: Executable filename - 'adobe.com'  
at 00003BBA: Executable filename - 'adobe.com'  
at 00003BF9: Executable filename - 'adobe.com'  
at 00003C28: Executable filename - 'adobe.com'  
at 00003C7F: Executable filename - 'adobe.com'  
at 00003CAD: Executable filename - 'adobe.com'
```

Figure 41: Listing of Embedded Domains - ticket\_354041.doc

The malware could also try to execute the Microsoft word application and this also provided us with validation that the file was written with Microsoft word.

```
at 00000000: office_magic_bytes - '\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00\x00'  
at 00026600: word_document - 'R\x00o\x00o\x00f\x00 \x...00\x00\x00\x00\x00\x00'  
at 00026780: word_document-W\x00o\x00r\x00d\x00D\x...00m\x00e\x00n\x00f\x00'  
at 0001FC83: word_document - 'MSWordDoc'
```

Figure 42: Listing of Installed Program Execution - ticket\_354041.doc

At this stage an assumption could be made by considering the results already found that the malware would execute on the targeted host and would try to contact the Adobe URLs in order to download, install and execute the Adobe application. That was not all, the following sections of the malware code required us to analyse the file even further, as indicated above that the file was packed.

## 5. Interesting words

The results from this section allowed us to make further assumptions about the intentions of the malware, the malware seemed to be either looking for two user accounts on the target host or could execute further using the accounts 'root' and / or 'pop'

```
at 00023768: Interesting keywords - 'root'  
at 0002246C: Interesting keywords - 'Pop'  
at 00027270: Interesting keywords - 'POP'
```

Figure 43: Listing of Interesting Words Found - ticket\_354041.doc

## 6. Ole Headers

The malware code consisted of the Office Object Linking, and Embedding (OLE) capability that could trick a user into enabling and downloading additional malicious content as previously observed from the embedded URLs.

```
at 00000000: Possible OLE2 header (e.g. MS Office documents) -  
'\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1'
```

Figure 44: Listing of OLE Headers - ticket\_354041.doc

The embedded objects are extracted to the user's temp directory where they are maybe launched with the default handler if the object within the document is clicked by the user. Once the document is closed the files are cleaned up by removing the temp files from the user's temp directory. While the document is opened, these files may be available to other processes on the system. The object embedded into the document from a file will make use of the Packager Object Server.

## 7. Macros embedded

This section of the code indicates that the malware is embedded with VBA macros that will be executed once the file runs on the target host.

```
at 00024C8C: Possible VBA macros - 'VBA'  
at 00024CB4: Possible VBA macros - 'VBA'
```

Figure 45: Listing of Indication of Existence of Macros - ticket\_354041.doc

We needed to understand the content and purpose of the VBA macros code found within the file as mentioned above. For the purpose of further analysis we first used the “oledump.py” to understand what the VBA macros code contained.

### Oledump.py Tool execution:

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
```

```
1: 113 '\x01CompObj'  
2: 4096 '\x05DocumentSummaryInformation'  
3: 4096 '\x05SummaryInformation'  
4: 4096 '1Table'  
5: 48951 'Data'  
6: 531 'Macros/PROJECT'  
7: 98 'Macros/PROJECTwm'  
8: M 11025 'Macros/VBA/ThisDocument'  
9: 7793 'Macros/VBA/_VBA_PROJECT'  
10: m 1153 'Macros/VBA/bebop'  
11: 837 'Macros/VBA/dir'  
12: M 13403 'Macros/VBA/uninstructed'  
13: 97 'Macros/bebop/\x01CompObj'  
14: 287 'Macros/bebop/\x03VBFrame'  
15: 98 'Macros/bebop/f'  
16: 112 'Macros/bebop/i01/\x01CompObj'  
17: 4396 'Macros/bebop/i01/f'  
18: 68 'Macros/bebop/i01/o'  
19: 0 'Macros/bebop/o'  
20: 52501 'WordDocument!'
```

Figure 46: Listing of VBA Macros Found - ticket\_354041.doc



The output results above indicated the multiple VBA macros that are embedded on the file as indicated from line item 1 to 3. The *oledump.py* tool has the ability to mark streams that contain VBA code. Output results of the given stream were viewed by adding “-s” with an object number. As we knew we were dealing with the VBA code the “-v” option would then instruct “*oledump.py*” to decompress the VBA code and make it easy to read. It is safe to say we found our malicious code based on the high level summary of the results below. Detailed results are available on appendix C.

```
Attribute VB_Name = "ThisDocument"  
Attribute VB_Base = "1Normal.ThisDocument"  
Attribute VB_GlobalNameSpace = False  
Attribute VB_Creatable = False  
Attribute VB_PredeclaredId = True  
Attribute VB_Exposed = True  
Attribute VB_TemplateDerived = True  
Attribute VB_Customizable = True
```

Figure 47: Listing of VBA Macros Attributes - ticket\_354041.doc

The “*officeparser.py*” tool allowed us to print similar information as *oledump.py*, however it would help the malware analysts with marking objects containing VBA code as indicated below.

#### Officeparser.py tool execution

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$  
officeparser.py --extract-macros ticket_354041.doc  
  
INFO: Saving VBA code to ./ThisDocument.cls  
INFO: Saving VBA code to ./uninstructed.bas  
INFO: Saving VBA code to ./bebop.frm
```

Figure 48: Listing of Extracted VBA Macros - ticket\_354041.doc

We then needed to analyse the VBA code further with the tools available within the REMnux analysis toolkit. The “*olevba.py*” tool performed all the steps of the process mentioned above including the basic analysis of the code that is embedded on the file. This provided us with the ability to be able to stretch the analysis of the VBA code even further. For the purpose of describing the analysis of the VBA code only, a summary of the code is provided and a detailed view of the code is available in appendix C.

#### Olevba.py tool execution:

remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples\$

olevba.py ticket\_354041.doc

olevba 0.51a - <http://decalage.info/python/oletools>

Flags      Filename

-----  
OLE:MAS-HB-- ticket\_354041.doc

---

---

FILE: ticket\_354041.doc

Type: OLE

-----  
VBA MACRO ThisDocument.cls

in file: ticket\_354041.doc - OLE stream: u'Macros/VBA/ThisDocument'

in file: ticket\_354041.doc - OLE stream: u'Macros/VBA/uninstructed'

-----  
BA MACRO bebop.frm

in file: ticket\_354041.doc - OLE stream: u'Macros/VBA/bebop'

-----  
(empty macro)

-----  
VBA FORM STRING IN 'ticket\_354041.doc' - OLE stream: u'Macros/bebop/i01/o'

-----  
**Lib "kernel32" Alias "HeapCreate"**

**Lib "user32" Alias "GetWindowText"**

**Lib "user32" Alias "EndDialog"**

**Function RtlAllocateHeap Lib "ntdll" (**

**Lib "kernel32" Alias "GetModuleHandle"**

**Lib "user32" Alias "GetDC"**

Figure 49: Listing of VBA Macros Code - *ticket\_354041.doc*

The output results above from the *olevba.py* tool provided us with the obfuscated code of the malware from the file that was analysed and the summary of results below provided us with an opportunity to further draw conclusions about the file being analysed. The listing above indicated VBA code from the "O" macro, the code indicated that the application will use windows functions "user32" and "kernel32" to execute some of the functions.

Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word or Publisher document is opened
Suspicious	RtlMoveMemory	May inject code into another process
Suspicious	Lib	May run code from a DLL
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)

Figure 50: Listing of Results Summary - ticket\_354041.doc

#### 4.2.1.1.1 Static Analysis Results Summary Overview

Figure 51, below a summary of the static analysis experiment conducted on the ticket\_354041.doc malware sample.

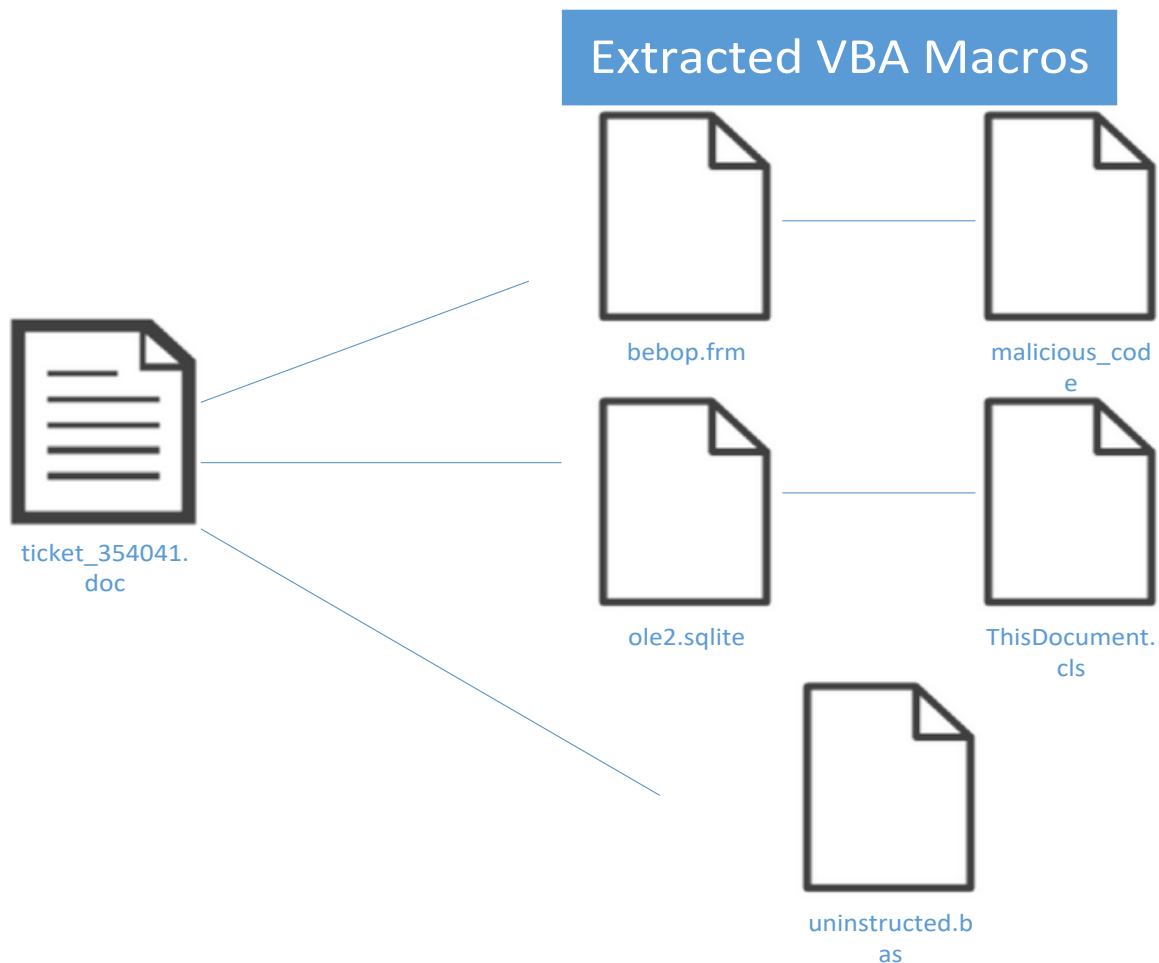


Figure 51: Ticker\_354041 Sample Overview ticket\_354041.doc

Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word or Publisher document is opened
Suspicious	RtlMoveMemory	May inject code into another process
Suspicious	Lib	May run code from a DLL
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)

Figure 52: Ticker\_354041 Sample Suspicious Capabilities ticket\_354041.doc

The malware artifact file analysed above allowed us to draw a conclusion that the file, if executed, can cause harm to a target system as the file consists of obfuscated code which, when executed, has the ability to access embedded URLs, execute portable executables, send emails and execute VBA scripts embedded within the code. Analysis of the file does not really provide us with true intention of the malware and this requires the file to be executed in an isolated environment to be able to observe the behaviour of the malware; Dynamic analysis provides us with this capability that is explained in detail in the following section.

#### 4.2.1.2 Dynamic Analysis

In this section of analysing the malware artifact, which is performing dynamic analysis on the malware artifact that was executed in the Cuckoo sandbox and its behaviour, was observed. A report with the results of the malware artifact was then generated and analysed in this section. The full report is made available in appendix D and in this section only a few areas from the report are discussed.

##### a) File Properties

The results in figure 53 below provided us with detailed results of the file itself and it was important to note that the hash values reported above are the same hash values as found when the file was statically analysed.

#### File Details

File name	ticket_354041.doc
File size	162816 bytes
File type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1251, Title: , Author: Laura, Template: Normal.dot, Last Saved By: Windows, Revision Number: 1, Name of Creating Application: Microsoft Office Word, Total Editing Time: 01:00, Create Time/Date: Wed Oct 19 14:33:00 2016, Last Saved Time/Date: Wed Oct 19 14:34:00 2016, Number of Pages: 1, Number of Words: 0, Number of Characters: 2, Security: 0
CRC32	28E780BB
MD5	543c0cf636bc0e56007e6211cd05ecf2
SHA1	400cb9f479fd5ab09aa895245e16ba999ce5142e
SHA256	3ea894203c48d37b73ce9202dec7eedbf1c724b707f7de058e42c18c3e55bd49
SHA512	909f4c09ee15da781a6405f806d5172d3a3e6f3e84d5e40df7b79885ffd76df989f44cae2587d3bb584af5e064280eebf70c0c2b9cc6edbe50dcfec7f316f330
Ssdeep	3072:TPzjPz+GMPyhgY0u7X6P2ab+PA5dIJ064tSk9qAERSEj7RdM+:TXzhgFPHasuJkQRLldM+
PEID	None matched
Yara	None matched
VirusTotal	VirusTotal lookup disabled, add your API key to the module

Figure 53: File Properties ticket\_354041.doc

#### b) Artifact Activity Screenshot

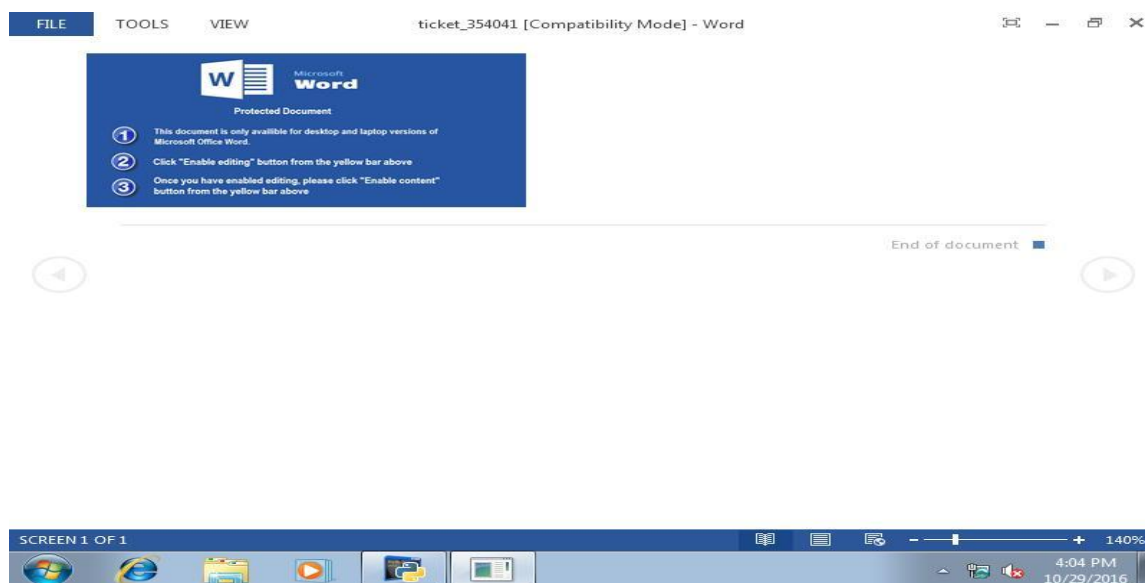


Figure 54: Activity Screenshot - ticket\_354041.doc

Malware artifact when it was executed within the Cuckoo sandbox, it then opened the word document as indicated above on figure 64.

#### c) Indicators of Compromise (IOC)

This part of the results provided us with indicators of compromise of what the actual artifact does on the system when it is executed and the behaviour or dynamic analysis of the artifact was observed.

The IOC section was broken down and explained in the following four areas:

### C.1. Signatures

<a href="#">raises_exception details</a>
<a href="#">dumped_buffer details</a>
<a href="#">allocates_rwx details</a>
<a href="#">creates_doc details</a>
<a href="#">creates_exe details</a>
<a href="#">memdump_urls details</a>
<a href="#">deletes_self details</a>
<a href="#">dropper details</a>
<a href="#">injection_runpe details</a>

Figure 55: File Signatures - ticket\_354041.doc

- **Raises exception** – indicate that one of more processes crashed
- **Dumped buffer** – One or more potentially interesting buffers were extracted, these generally inject code, configuration data, etc.
- **Allocates\_rwx** – Allocates read-write-execute memory and to usually unpack itself
- **Creates\_exe** - Creates executable files on the file system
- **Memdump\_URLs** – Potentially malicious URLs were found in the process memory dump
- **Deletes\_self** – Deletes its original binary from disk
- **Dropper** – Drops a binary and executes it
- **Injection\_runpe** - Executed a process and injected code into it, probably while unpacking

### C.2. Dropped Files

---

[7a88bf375bb95df5\\_msforms.exd](#)  
[4826c0d860af884d\\_~wrs{4531b2a9-06c8-4551-89f2-a3095586aa32}.tmp](#)  
[a1272deb82ce95c1\\_ge443.exe](#)  
[548b54a29f9de180\\_~\\$normal.dotm](#)  
[30ac5dae441a008f\\_~\\$cket\\_354041.doc](#)

Figure 56: Files dropped by the Malware - ticket\_354041.doc

The malware artifact drops the five files once it has executed and most importantly the artifact also dropped an executable file “a1272deb82ce95c1\_ge443.exe” as indicated above.

### C.3. Behaviour Summary

```
File-Written
• C:\Users\root\AppData\Local\Temp\~$cket_354041.doc
• C:\Users\root\AppData\Roaming\Microsoft\Templates\~$Normal.dotm
• C:\Users\root\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\~$WRS(4531B2A9-06C8-4551-89F2-A3095586AA32)\~$a1272deb82ce95c1_ge443.exe
• C:\Users\root\AppData\Local\Temp\VBEMForms.exe
• C:\Users\root\AppData\Local\Temp\VBEMForms.exd
```

Figure 57: File Written - ticket\_354041.doc

The malware then writes itself into the above indicating directories on figure 67 and also drops the executable ge443.exe that we also see from the dropped file section above.

```
File-Opened
• C:\Windows\Globalization\Sorting\sortdefault.nls
• C:\Users\root\Favorites\desktop.ini
• C:\Users\root\AppData\Local\Temp
• C:\Windows\Fonts\arial.ttf
• C:\
• C:\Windows\System32\spool\drivers\color\sRGB Color Space Profile.icm
• C:\Windows\System32\en-US\tzres.dll.mui
• C:\Windows\System32\en-US\user32.dll.mui
• C:\Users\root\AppData\Roaming
• C:\Program Files\Microsoft Office\Office15\WMLIB.DLL
• C:\Users\root\AppData\Local\Temp\VBEMForms.exe
• C:\Program Files\Microsoft Office\Office15\
• C:\Program Files\Common Files\Microsoft Shared\OFFICE15\Cultures\OFFICE.OOF
• C:\Users\root\Contacts\desktop.ini
• C:\Windows\System32\en-US\SETUPAPI.dll.mui
• C:\Users\root\AppData\Roaming\Microsoft\Word\STARTUP\
• C:\Windows\System32\sspicli.dll
• C:\Users\root\AppData\Roaming\Microsoft\
• C:\Windows\System32\uxtheme.dll
• C:\Users\root\AppData\Roaming\Microsoft\Templates\
• C:\Windows\Microsoft.NET\Framework\v2.0.50727\mscorlib.dll
• C:\Users\root\AppData\Local
• C:\Users\root\Application Data\Microsoft\Forms\WINWORD.box
• C:\Windows\System32\en-US\setupapi.dll.mui
```

Figure 58: File opened - ticket\_354041.doc

We saw in figure 58 that the artifact then opened and executed in the above directly, and most importantly it used of the Windows functions “user32.dll” that is explained in detail in the literature review and also in Appendix C.

```
Registry Key-Read
• HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize\DisableMetaFiles
• HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Nls\CustomLocale\en-US
• HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Nls\ExtendedLocale\en-US
```

Figure 59: Registry Keys Read - ticket\_354041.doc

The artifact then read three registry keys and one of the registry keys allowed the malware to Disable Metafiles on the system as indicated on figure 59 above.

## C.4. Processes

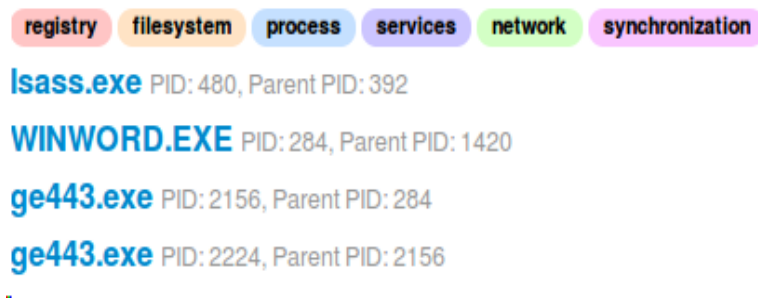


Figure 60: Process Executed ticket\_354041.doc

In figure 60 we saw the malware artifact running four processes on the system and the most common was the executable file that was first dropped on some of the directories mentioned above.

### 4.2.1.2.1 Results Overview Summary

Figure 61, below is a graphical view of the behaviour results of the malware sample.

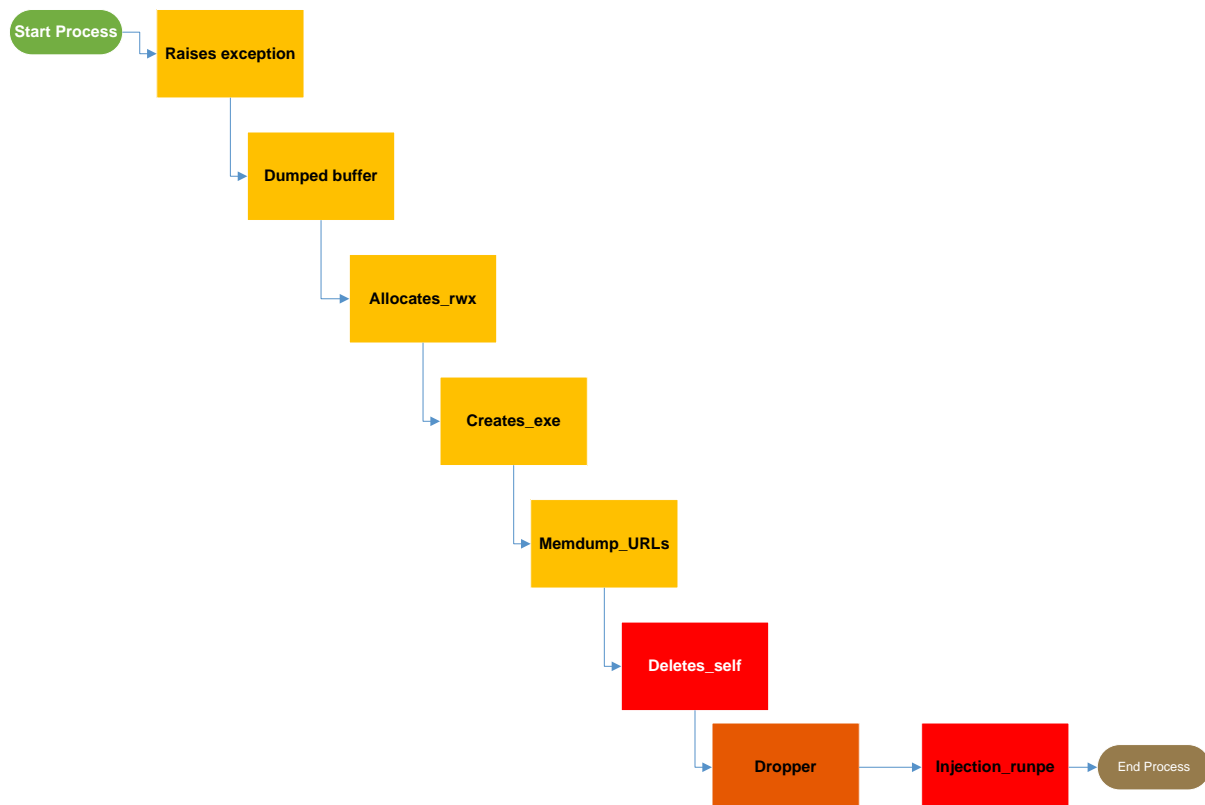


Figure 61: Results Overview

### 4.2.1.3 Virus Total Results

This is the last step of the analyses process to confirm if the malware artifact is in fact a malware artifact. This process was achieved by running the malware artifact through the online



Antivirus scan engine that connects to a maximum of 55 well known antivirus engines. The results from virus total indicated that the artifact that was analysed was classified as a Trojan malware. Only 36 out of 55 antivirus engines were able to determine and confirm that the artifact was indeed a malware file. Figure 62 below provides us with the results from Virus Total and also provided the name of the malware artifact that was analysed further, using static and dynamic analysis techniques as mentioned above.

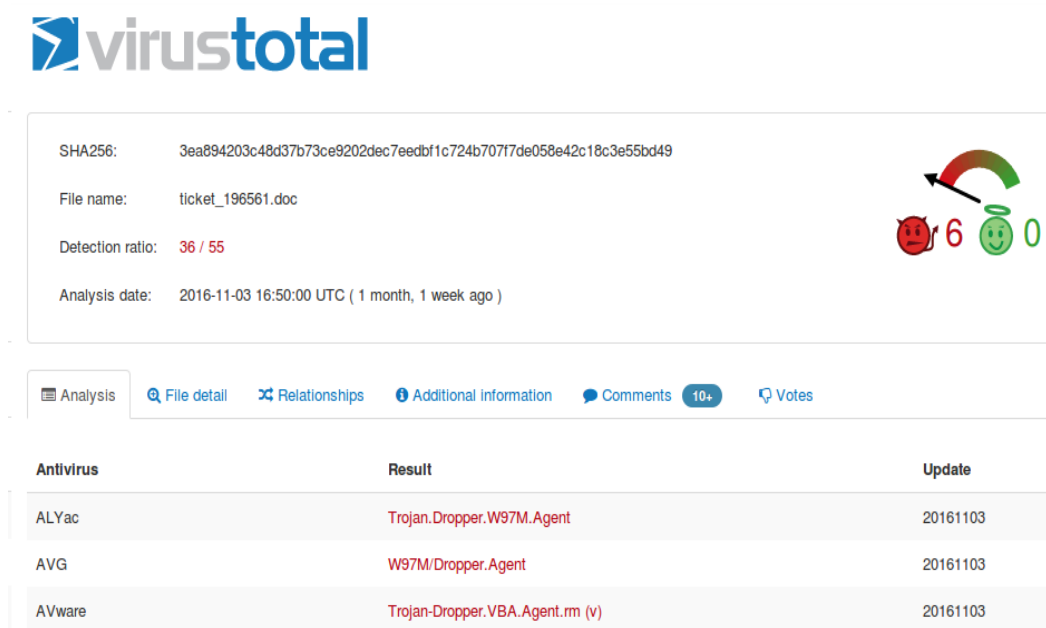


Figure 62: Virus Total Results - ticket\_354041.doc

## 4.2.2 Analysis of 1123211 – 090SD.exe

### 4.2.2.1 Static Analysis

Before the analysis of the malware artifact acquired could commence, it was important to fingerprint the file, and confirm the type of file and properties of the artifact in order to determine the type of file that was analysed.

#### a) File Fingerprint

The first step was to uniquely identify the malware artifact in order to be able to fingerprint or identify the malicious artifact in future. MD5, SHA1 and SHA256 sums of the artifact were created and these hashes themselves were enough to be able to fingerprint the file. This part of the process was achieved by running the artifact through the “*rhash*” tool that is a console utility for computing and verifying hash sums of files. By executing the *rhash* command with a combination of –M (for MD5) or –H (SHA1) or –sha256 hash values of file were provided. The results of the tool are provided below:

### **rhash tool execution:**

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ rhash -M
1123211-090SD.exe
17a9f0b4bf1cfdd20a492da6620353aa 1123211-090SD.exe
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ rhash -H
1123211-090SD.exe
baf030ac03afef1a0deb3f2b01d4056153732aed 1123211-090SD.exe
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ rhash --
sha256 1123211-090SD.exe
f9460df7c69640e717c32b9aae57473e4a17d8e049df23d3392034cb039fb6e7      1123211-
090SD.exe
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$
```

Figure 63: Listing of File Section Hashes – 1123211-090sd.exe

The output results above indicate that the artifact's MD5, SHA1 and SHA256 hashes are as follows:

MD5 = 17a9f0b4bf1cfdd20a492da6620353aa

SHA1 = baf030ac03afef1a0deb3f2b01d4056153732aed

SHA256 = f9460df7c69640e717c32b9aae57473e4a17d8e049df23d3392034cb039fb6e7

This hash was later confirmed from the other tools that were used as part of performing static analysis.

### **b) File Details**

The second setup was to then confirm the properties and type or format of the file that we were using. The process started by first executing the “file” command against the '1123211-090SD.exe' artifact. The results of the command are as follows:

#### **File command execution:**

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ file
1123211-090SD.exe
1123211-090SD.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS
Windows
```

Figure 64: Listing of File Details - 1123211-090sd.exe

The results above indicated that the format of the file that would be analysed further was an executable file.

### c) File Signature

The third step of the analysis process was to determine the malware signatures of the file by running the “Clamscan” tool against the file.

#### Clamscan tool execution:

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$  
clamscan 1123211-090SD.exe
```

#### ----- SCAN SUMMARY -----

```
Known viruses: 3832461  
Engine version: 0.98.7  
Scanned directories: 0  
Scanned files: 1  
Infected files: 0  
Data scanned: 0.46 MB  
Data read: 0.45 MB (ratio 1.01:1)  
Time: 5.819 sec (0 m 5 s)
```

Figure 65: Listing of File Signature - 1123211-090sd.exe

The results from the Clamscan output indicated that the file is not infected or is a malware but again this could be a false negative; further analysis was required.

### d) Indicators of Compromise (IOC)

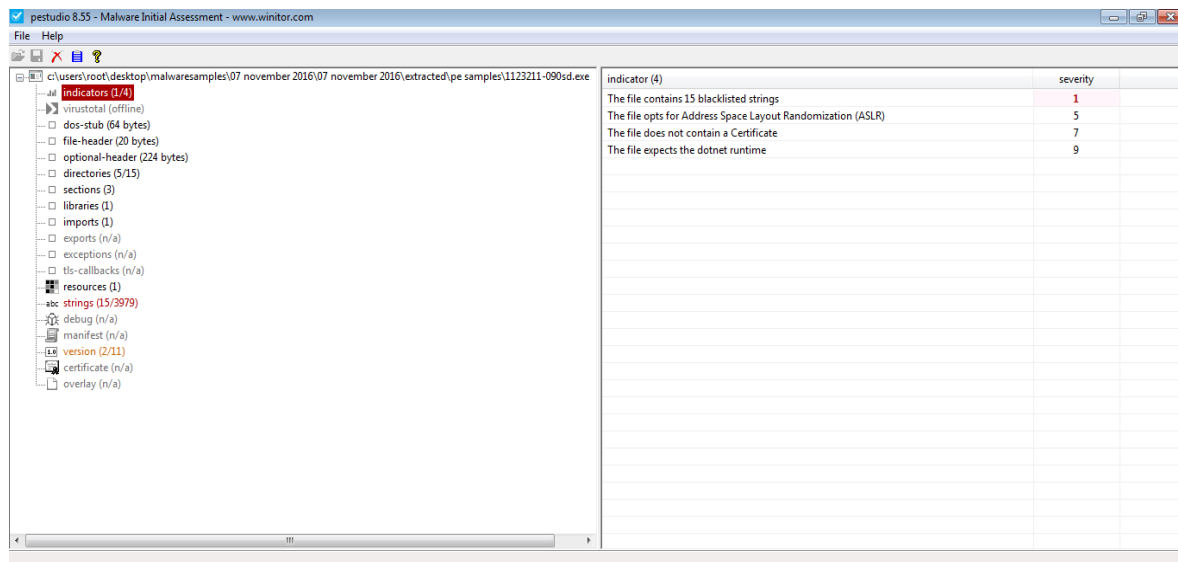


Figure 66: Indicators of Compromise – 1123211-090sd.exe

The results from the above figure 66 indicated on 1 out 4 IOCs and these IOCs were the 15 blacklisted strings that were found within the malware artifact.

### e) Strings

PEStudio Output results

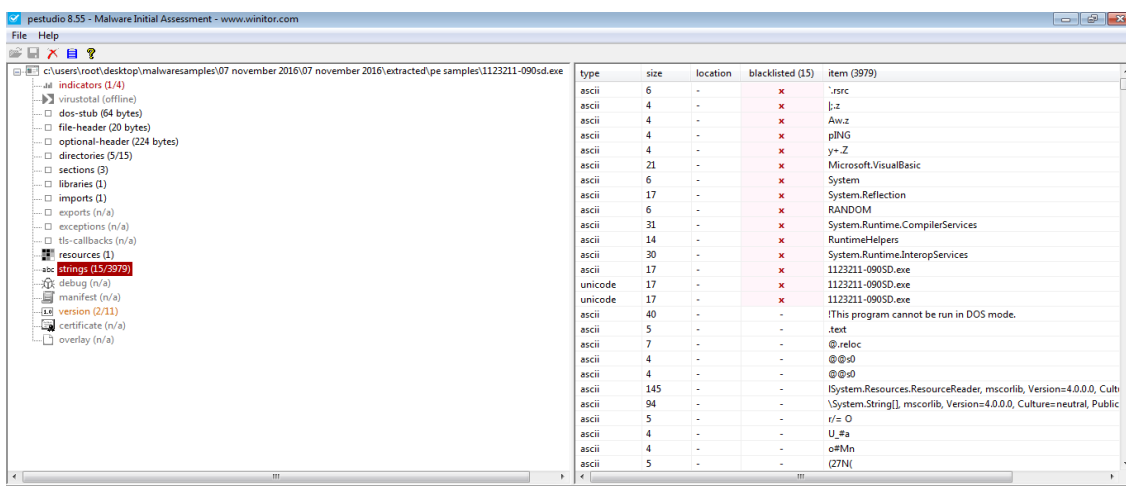


Figure 67: Strings – 1123211-090sd.exe

Figure 67 above indicated which of the 15 blacklisted strings were detected as the IOCs.

### String Offset from the artifact

Searching through the strings was a simple way to get hints about the functionality artifact.

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$
pestr -n 1123211-090SD.exe
```

```
ISystem.Resources.ResourceReader, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089#System.Resources.RuntimeResourceSet
```

```
0x70af4  get_Computer
0x70b01  m_ComputerObjectProvider
0x70b1a  get_Application
0x70b2a  m_AppObjectProvider
0x70b3e  User
0x70b43  get_User
0x70b4c  m_UserObjectProvider
-----
0x715cb  AppDomain
0x715d5  get_CurrentDomain
0x715e7  MethodInfo
-----
0x71745  1123211-090SD
0x71753  1123211-090SD.exe
0x71772  GetObject
0x7178e  invoke
0x7179c  Load
-----
0x71a7c  4System.Web.Services.Protocols.SoapHttpClientProtocol
0x71ab2  Create_Instance
0x71ac5  Dispose_Instance
```

Figure 68: Listing of String Offsets - 1123211-090sd.exe

Listing 20 indicated that there were several windows functions that the malware would use when executed and only 7 of these functions are provided. These functions as observed include the following:

- Get\_Computer
- Get\_Application
- Get\_User
- Get\_CurrentDomain
- GetObject
- Create\_Instance
- Dispose\_Instance

These functions indicate that the malware will attempt to get the name of the computer that it is running on, execute one or more applications, find the user that is currently logged on the affected system and also check if the affected system is part of a domain. The malware will then try to get one or more objects, create further instances and then try to delete those created instances in order to hide its tracks.

#### f) Extracted Code - Packed

Extracted and decoded suspicious patterns from malicious file using the “*balbuzard.py*” tool

#### Balbuzard.py tool execution:

File: 1123211-090SD.exe

File type according to magic: MS Windows PE

```
at 0000004E: EXE PE DOS message - 'This program cannot be run in DOS mode'
at 00070C56: Executable filename - 'user32.dll'
at 0007175B: Executable filename - '090SD.exe'
at 00071D2E: Executable filename - 'mscoree.dll'
at 00000178: EXE: section name - '.text'
at 000001A0: EXE: section name - '.rsrc'
at 000001C8: EXE: section name - '.reloc'
at 00071114: Interesting keywords - 'POP'
```

Figure 69: Listing of Extracted Code - 1123211-090sd.exe

The above output results found three files as outlined below; this could mean that these files could be either be dropped or executed on an infected system. The DLL files are important Windows functions as explained in Appendix C and this could mean that the malware could execute or use these functions and the executable to perform further actions on the infected system.

Additional Files found from the results above:

1. User32.dll - Important Windows Function

2. 090SD.exe - Executable
3. Mscoree.dll - Important Windows Function

### Pedump tool execution

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ pedump 1123211-090SD.exe
```

The results below indicate that the packer used is a Microsoft Visual C# and we once again saw the functions (CorExeMain) and module (mscoree.dll) that could be imported.

=== Packer / Compiler ===

MS Visual C# / Basic .NET

Figure 70: Listing of Packer PEDUMP - 1123211-090sd.exe

### g) Examined a PE file and detected suspicious characteristics - Entropy

Used the “pescanner” tool to examine and to detect suspicious characteristics from the malware artifact.

#### Pescanner tool execution

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ pescanner '1123211-090SD.exe'
```

### Sections

Name	VirtAddr	VirtSize	RawSize	MD5	Entropy	
.text	0x2000	0x70d44	0x71000	6232066005e8bc55677d912943586130	7.729978	[SUSPICIOUS]
.rsrc	0x74000	0x2b8	0x1000	68ef18aad256956be91563b7bc17d78c	0.716118	[SUSPICIOUS]
.reloc	0x76000	0xc	0x1000	d507f949a165f0c27c241348e0b96368	0.016408	[SUSPICIOUS]

Figure 71: Listing of File Entropy - 1123211-090sd

The .text section indicated a high Entropy value of 7.72. Entropy is used in different ways by malware authors and analysts that provides a rough estimation of whether the file is encrypted or not and it helps to determine if further analysis is required.

### h) Imported/Exported Functions

#### Pedump tool execution

```
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$ pedump 1123211-090SD.exe
```

Once again the functions (CorExeMain) and module (mscoree.dll) that could be imported were observed from the results of these tools which still validates the already found results, that indicated that the artifact will perform certain imports when executed.

```

=== IMPORTS ===
MODULE_NAME      HINT  ORD  FUNCTION_NAME
mscoree.dll      0     _CorExeMain

```

Figure 72: Listing of IMPORTS PEDUMP- 1123211-090sd.exe

### i) Additional Packed and Extracted Properties

Results from this section were extracted and compared between the following tools that were used from the REMnux and Window 7 systems respectively. Detailed results are available in appendix E and only compared sections are discussed in this section as these tools produced similar results when executed.

Tools used in this section are:

- PEDUMP
- CFF Explore
- PEFRAME

### I.1 Pedump Tool Output

The “pedump” tool is similar to the “pescanner” tool as both these tools are used to analyse portable executables. The pedump tools output more information than the pescanner tool.

#### Pedump tool execution

```

remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples$
pedump 1123211-090SD.exe

```

#### === DOS STUB ===

The results below indicated the obfuscated area of the malware and the malware artifact will not execute in DOS mode.

```

00000000: 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 |.....!..L.!Th|
00000010: 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f |is program canno|
00000020: 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 |t be run in DOS |
00000030: 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 |mode....$......|

```

Figure 73: Listing of DOS STUB - 1123211-090sd.exe

#### === DATA DIRECTORY ===

The output results below indicated that the characteristics of the executable would run on a 32bit Windows system and that the executable is a terminal server, as indicated in the characteristics DLL section from the figure below.

```

signature:                "PE\x00\x00"

# IMAGE_FILE_HEADER:
    Machine:                332            0x14c    x86
    NumberOfSections:       3              3
    TimeDateStamp:          "2016-09-14 21:51:48"
    PointerToSymbolTable:    0              0
    NumberOfSymbols:         0              0
    SizeOfOptionalHeader:   224            0xe0
    Characteristics:        258            0x102    EXECUTABLE_IMAGE, 32BIT_MACHINE

# IMAGE_OPTIONAL_HEADER32:
    Magic:                   267            0x10b    32-bit executable
    LinkerVersion:           8.0
    SizeOfCode:               462848         0x71000
    SizeOfInitializedData:    8192         0x2000
    SizeOfUninitializedData:  0              0
    AddressOfEntryPoint:     470334         0x72d3e
    BaseOfCode:               8192         0x2000
    BaseOfData:               475136         0x74000
    ImageBase:                4194304        0x400000
    SectionAlignment:         8192         0x2000
    FileAlignment:            4096         0x1000
    OperatingSystemVersion:   4.0
    ImageVersion:              0.0
    SubsystemVersion:         4.0
    Reserved1:                 0              0
    SizeOfImage:              491520         0x78000
    SizeOfHeaders:            4096         0x1000
    CheckSum:                  0              0
    Subsystem:                 2              2    WINDOWS_GUI
    DllCharacteristics:        34112         0x8540    DYNAMIC_BASE, NX_COMPAT, NO_SEH
    SizeOfStackReserve:       1048576        0x100000
    SizeOfStackCommit:        4096         0x1000
    SizeOfHeapReserve:        1048576        0x100000
    SizeOfHeapCommit:         4096         0x1000

```

Figure 74: Listing of File Directory Listing - 1123211-090sd.exe

### I.2 CFF Explore Tool Output

The output results on figure 75 below from the CFF Explore tool validate the results found in figure 74 above.

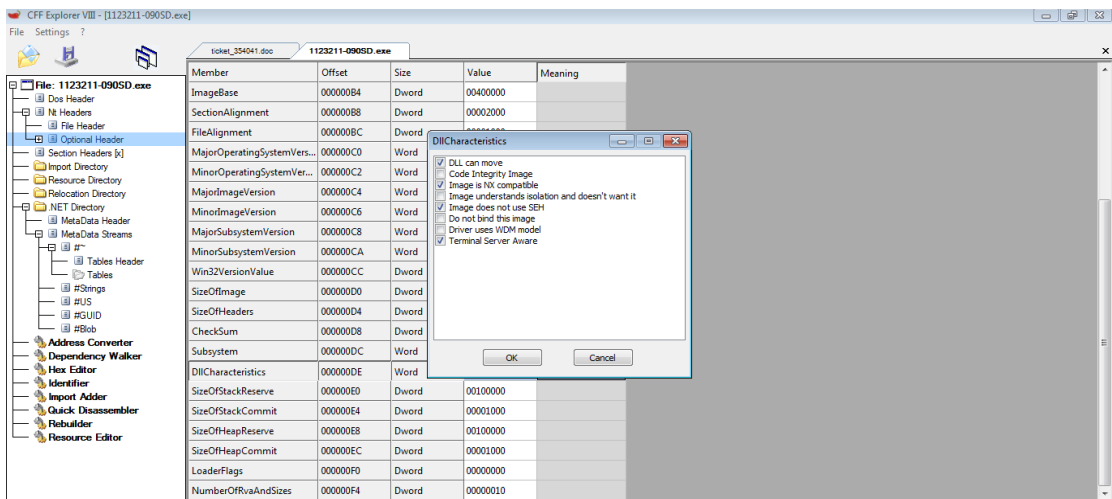


Figure 75: Optional Headers – 1123211-090sd.exe

### I.3 PEFRAME TOOL

The PEFRAME toll was executed and the following results were outputted by the tool.



remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/PE Samples\$  
peframe 1123211-090SD.exe

### I.3.1 Packers Found

#### Packer matched [4]

```
-----  
Packer      Microsoft Visual C# / Basic .NET  
Packer      Microsoft Visual Studio .NET  
Packer      .NET executable  
Packer      Microsoft Visual C# v7.0 / Basic .NET
```

Figure 76: Listing of Packer PEFRAME Results- 1123211-090sd.exe

The results from figure 76 indicated that there were actually four packed files as opposed to one file that was detected in the earlier section of packed files found.

### I.3.2 Suspicious Sections

#### Suspicious Sections discovered [3]

```
-----  
Section     .text  
Hash MD5    6232066005e8bc55677d912943586130  
Hash SHA-1  406c1d93ac94f5b6d09d1917b1727220bd7d1fc5  
Section     .rsrc  
Hash MD5    68ef18aad256956be91563b7bc17d78c  
Hash SHA-1  c038e96741975febd9c58e92a099c568072ffc15  
Section     .reloc  
Hash MD5    d507f949a165f0c27c241348e0b96368  
Hash SHA-1  9f42c410b7b7ea4dbbabec2f21b965afe6954340
```

Figure 77: Listing of Suspicious Sections PEFRAME Results- 1123211-090sd.exe

Figure 77 from the PEFRAME results indicated that three sections and their hash values were found within the malware that were the following:

```
.text  
.rsrc  
.reloc
```

### I.3.3 Discovered Files

#### File name discovered [3]

```
-----  
Executable  1123211-090SD.exe  
Library     mscoree.dll  
Library     user32.dll
```

Figure 78: Listing of Packer PEFRAME Results- 1123211-090sd.exe

Figure 78 from the PEFRAM results indicated that there were two library files that could be used by the malware if executed, being mscoree.dll and user32.dll.

### j) Dependency Walker

The mscorre.dll and kernel32.dll were observed from figure 79 below to have had dependency DLL or function files required by the malware to be able to use important Windows functions in order to execute properly on the affected host in addition to the user32.dll observed from the results provided in the earlier sections.

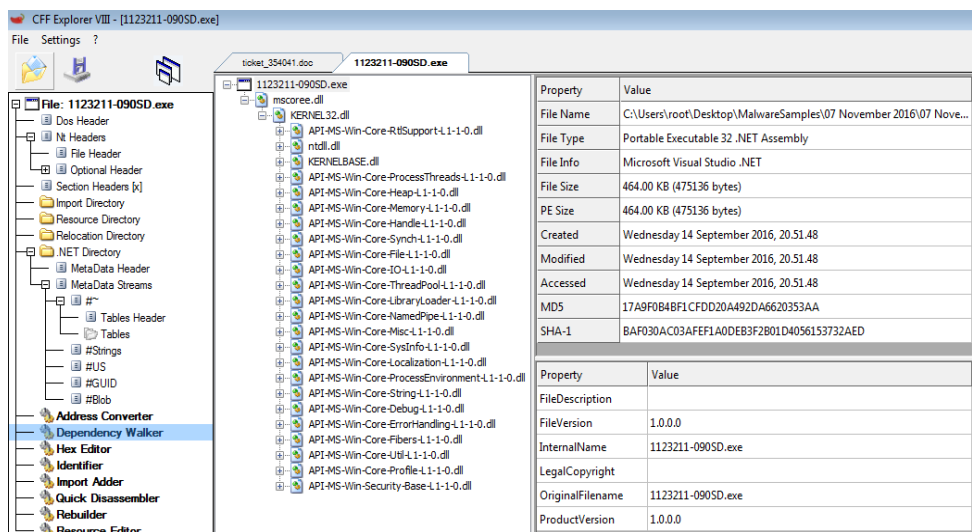


Figure 79: Dependency Walker – 1123211-090sd.exe

#### 4.2.2.1.1 Static Analysis Results Summary Overview

Figure 80, provides a summary overview of the results for the executable malware sample.

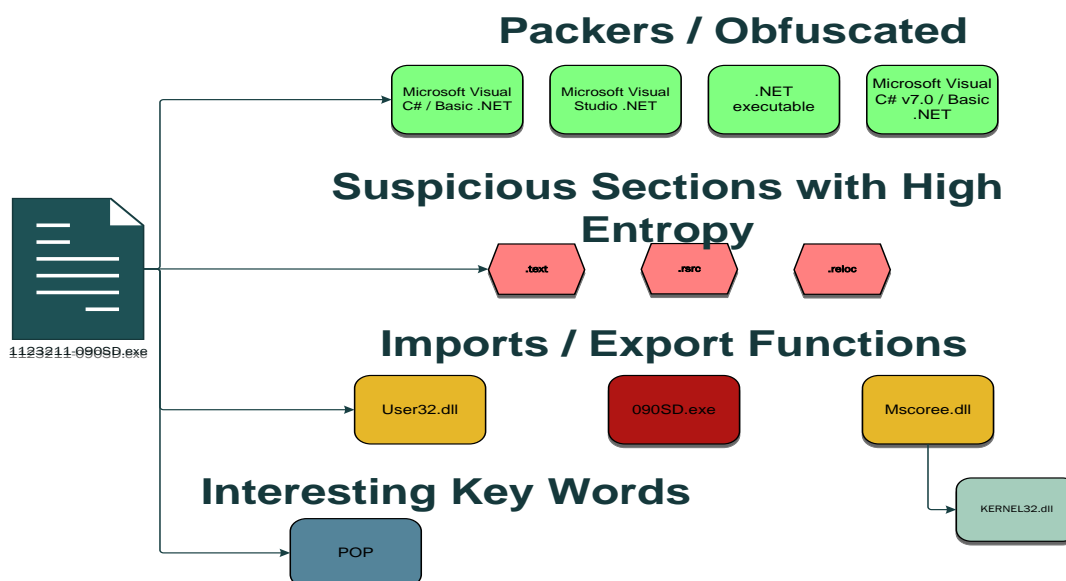


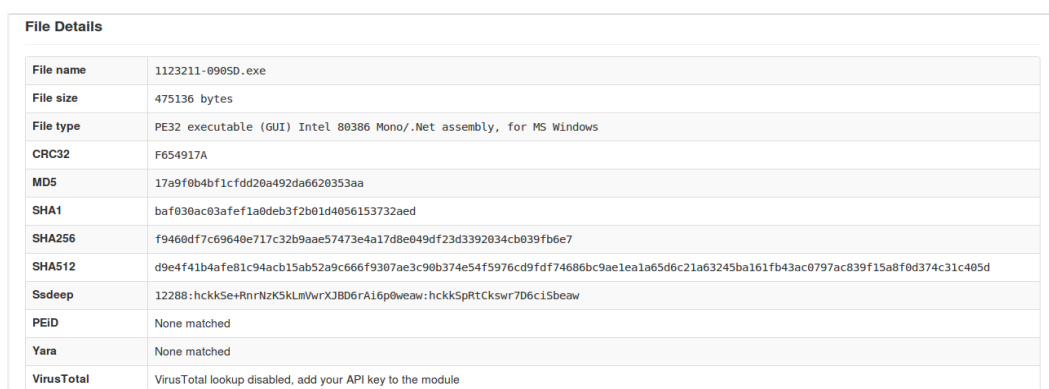
Figure 80: Static Analysis Results Overview - 1123211-090SD.exe

### 4.2.2.2 Dynamic Analysis

In this section of analysing the malware artifact which is performing dynamic analysis on the malware artifact that was executed in the Cuckoo sandbox and its behaviour, was observed. A report with the results of the malware artifact was then generated and analysed in this section. The full report is made available in appendix F and in this section only a few areas from the report are discussed.

#### a) File Properties

The results below provided us with detailed results of the file itself and it was important to note that the hash values reported above are the same hash values as found when the file was statically analysed.



File Details	
File name	1123211-090sd.exe
File size	475136 bytes
File type	PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
CRC32	F654917A
MD5	17a9f0b4bf1cfdd20a492da6620353aa
SHA1	baf030ac03afef1a0deb3f2b01d4056153732aed
SHA256	f9460df7c69640e717c32b9aae57473e4a17d8e049df23d3392034cb039fb6e7
SHA512	d9e4f41b4afe81c94acb15ab52a9c666f9307ae3c90b374e54f5976cd9fd74688bc9ae1ea1a65d6c21a63245ba161fb43ac0797ac839f15a8f0d374c31c405d
Sdeep	12288:hckk5e+RnrNzK5kLmWvrXJB06rA16p0weaw:hckkSprtCkswr7D6c1Sbeaw
PEID	None matched
Yara	None matched
VirusTotal	VirusTotal lookup disabled, add your API key to the module

Figure 81: File Details – 1123211-090sd.exe

#### b) Indicators of Compromise (IOC)

This part of the results provided us with indicators of compromise of what the actual artifact does on the system when it is executed, and the behaviour or dynamic analysis of the artifact was observed.

**The IOC section was broken down and explained in the following four areas:**

##### b.1 File Signatures

Signatures	
<a href="#">antivm_queries_computername details</a>	
<a href="#">antivm_memory_available details</a>	
<a href="#">raises_exception details</a>	
<a href="#">dumped_buffer details</a>	
<a href="#">network_bind details</a>	
<a href="#">allocates_rwx details</a>	
<a href="#">infostealer_browser details</a>	
<a href="#">recon_checkip details</a>	
<a href="#">antivm_network_adapters details</a>	
<a href="#">packer_entropy details</a>	
<a href="#">memdump_urls details</a>	
<a href="#">antisandbox_sleep details</a>	
<a href="#">persistence_autorun details</a>	
<a href="#">antiav_avast_libs details</a>	
<a href="#">infostealer_bitcoin details</a>	
<a href="#">deletes_self details</a>	
<a href="#">has_wmi details</a>	
<a href="#">infostealer_im details</a>	
<a href="#">injection_thread details</a>	
<a href="#">infostealer_mail details</a>	
<a href="#">stealth_hiddenfile details</a>	
<a href="#">injection_runpe details</a>	

Figure 82: File Signatures – 1123211-090sd.exe

- **Antivm\_queries\_computername** – Queried for the computer name
- **Antivm\_memory\_available** – Checked the amount of memory in system; this could be used to detect virtual machines that have a low amount of memory available
- **Raises\_exception** – One of more processes crashed
- **Dumped\_buffer** – One or more potentially interesting buffers were extracted; these generally contained injected code, configuration data, etc.
- **Network\_bind** – Started servers listening
- **Allocates\_rwx** - Allocated read-write-execute memory (usually to unpack itself)
- **Infostealer\_browser** - Stole private information from local internet browsers
- **Recond\_checkip** – Looked up the external IP address
- **Antivm\_network\_adapters** – Checked adapters’ addresses that were used to detect virtual network interfaces
- **Packer\_entropy** – The binary likely contained encrypted or compressed data
- **Memdump\_urls** - Potentially malicious URLs were found in the process memory dump
- **Antisandbox\_sleep** – A process attempted to delay the analysis task
- **Persistence\_autorun** – Installed itself for autorun at Windows startup
- **Antiav\_avast\_libs** – Tried to detect Avast Antivirus through the presence of a library
- **Infostealer\_bitcoind** – Attempted to access Bitcoin / ALTCoin wallets
- **Deletes\_self** - Deleted its original binary from disk
- **Has\_wmi** – Executed one or more WMI queries

- **Infostealer\_mail** – Harvested information related to installed instant messenger clients
- **Stealth\_hiddenfile** - Attempted to modify Explorer settings to prevent hidden files from being displayed
- **Injection\_runpe** – Executed a process and injected code into it, probably while unpacking

## b.2 Dropped Files



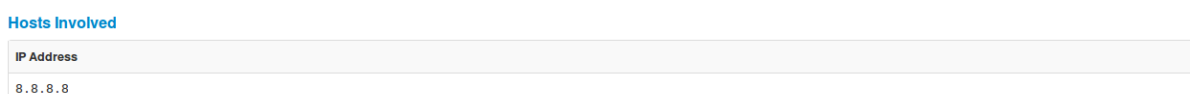
A screenshot showing four text files listed in a blue font. The files are: cca4a46fb8ca99e5\_pidloc.txt, e3b0c44298fc1c14\_holdermail.txt, 085bcb597bbd610a\_pid.txt, and b3d510ef04275ca8\_holderwb.txt.

Figure 83: Dropped Files – 1123211-090SD.exe

The malware artifact drops the four text files, as indicated in figure 83, once it has executed and most importantly the artifact also dropped text file as indicated above that could have contained email details. This could have meant that the malware could use those details to send out back to the command server.

## b.3 Network Analysis

This section provided us with the results of the communication attempt performed by the malware.



A screenshot of a table titled "Hosts Involved". The table has a header row with "IP Address" and a single data row with the value "8.8.8.8".

IP Address
8.8.8.8

Figure 84: Hosts Involved – 1123211-090SD.exe

Figure 84 above provided us with the IP or DNS server that the malware was using to communicate. The IP address entry is the DNS settings that were configured on our analysis sandbox system within the Cuckoo system.

#### DNS Requests

Domain	IP Address
teredo.ipv6.microsoft.com	
www.microsoft.com	88.221.244.240
cr1.microsoft.com	196.14.9.35
fe2.update.microsoft.com	134.170.58.189
dns.msftncsi.com	131.107.255.255
www.download.windowsupdate.com	13.107.4.50
ds.download.windowsupdate.com	196.14.9.32
whatismyipaddress.com	23.42.12.58
mail.humecmboard.com.my	124.217.254.9
download.microsoft.com	184.29.108.240
www.msftncsi.com	196.26.223.26

Figure 85: DNS Requests– 1123211-090SD.exe

Figure 85 above provided us with 11 Domains and IP addresses that the malware was using to communicate. The results above provided two domains below:

- Whatismyipaddress.com
- Mail.humecmboard.com.my

The two domains mentioned indicate that the malware tried to first look up what the IP address of the host was to see if it is not using an IP address within a sandbox environment and it then tried to communicate with an email hosting domain. This also validated the text file that was dropped in figure 83 that indicated that the text file could have contained email details that were used by the malware.

#### b.4 Behaviour Summary

This section provides us with a summary of the behaviour of the malware that is in support of the file signature results already discussed within the IOC section. Out of the 10 behaviour results available, only 5 areas of behaviour are discussed in this section.

```
File-Read
• C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config
• C:\Users\root\AppData\Local\Temp\holderwb.txt
• C:\Users\root\AppData\Local\Microsoft\Windows Mail\account{F94F87C6-16E7-4260-898A-F914CCBA59C5}.oeaccount
• C:\Users\root\AppData\Local\Microsoft\Windows Mail\account{B080C101-78A1-4021-B0C9-74E9BA614F9C}.oeaccount
• C:\Users\root\AppData\Local\Microsoft\Windows Mail\account{F096F190-3EAE-4F4B-8D9E-B38FCCB0CE0}.oeaccount
• C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config
• C:\Users\root\AppData\Local\Microsoft\Windows\History\Low\History.IE5\MSHist012016102820161029\index.dat
• C:\Users\root\AppData\Local\Microsoft\Windows\History\Low\History.IE5\index.dat
• C:\Users\root\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012016110820161109\index.dat
• C:\Users\root\AppData\Local\Microsoft\Windows\History\History.IE5\index.dat
• C:\Users\root\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012016102420161031\index.dat
```

Figure 86: File Read – 1123211-090SD.exe

The results from the File-Read behaviour in figure 86 above indicated that the malware read one of the text files that was the “holderwb.txt” that it dropped earlier. We also observed

“c:\users\root\AppData\Local\Microsoft\Windows Mail” directories that were read by the malware.

```
File-Written
• C:\Users\root\AppData\Roaming\pid.txt
• C:\Users\root\AppData\Roaming\pidloc.txt
• C:\Users\root\AppData\Local\Temp\holderwb.txt
```

Figure 87: File Written– 1123211-090SD.exe

The File-Written behaviour in figure 87 above indicated that the malware wrote the three text files that were discovered earlier into these three directories.

```
File-Deleted
• C:\Users\root\AppData\Local\Temp\holdermail.txt
• C:\Users\root\AppData\Local\Temp\holderwb.txt
• C:\Users\root\AppData\Roaming\Microsoft\CLR Security Config\v2.0.50727.312\security.config.cch.1364.23315033
• C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\enterprisesec.config.cch.1364.23314993
• C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\security.config.cch.1364.23314993
```

Figure 88: File Deleted – 1123211-090SD.exe

The File-Deleted behaviour in figure 88 indicated that the malware then deleted itself from the five directories as indicated.

**u**

Figure 89: File Opened – 1123211-090SD.exe

The File-opened behaviour in figure 89 produced over 20 directories that were accessed by the malware, the interesting directory that observed which one contained the “mscorrc.dll”. This DLL was detected to be one of the imports done by the malware as discovered in the Static analysis performed on this malware.

```
File-Copied
• C:\Users\root\AppData\Local\Temp\1123211-090SD.exe -> C:\Users\root\AppData\Roaming\WindowsUpdate.exe
```

Figure 90: File Copied – 1123211-090SD.exe

The File-Copied behaviour in figure 90 indicated that the malware copied itself into the “c:\users\root\AppData\Roaming\WindowsUpdate.exe” directory. This behaviour could also mean that the malware could have renamed itself and replaced itself to be the “WindowsUpdate.exe” application and this could also be a form of communication attempt that the malware performed.

## b.5 Processes



Figure 91: Processes – 1123211-090SD.exe

The output results from figure 91 above indicate that the malware executed only 5 processes. Process “lsass.exe” and “vbc.exe” are additional processes that could have been dropped by the malware as they could have obfuscated within the original file.

### Results Overview Summary

Figure 92, below provides a summary of the behaviour of the malware sample that was analysed.

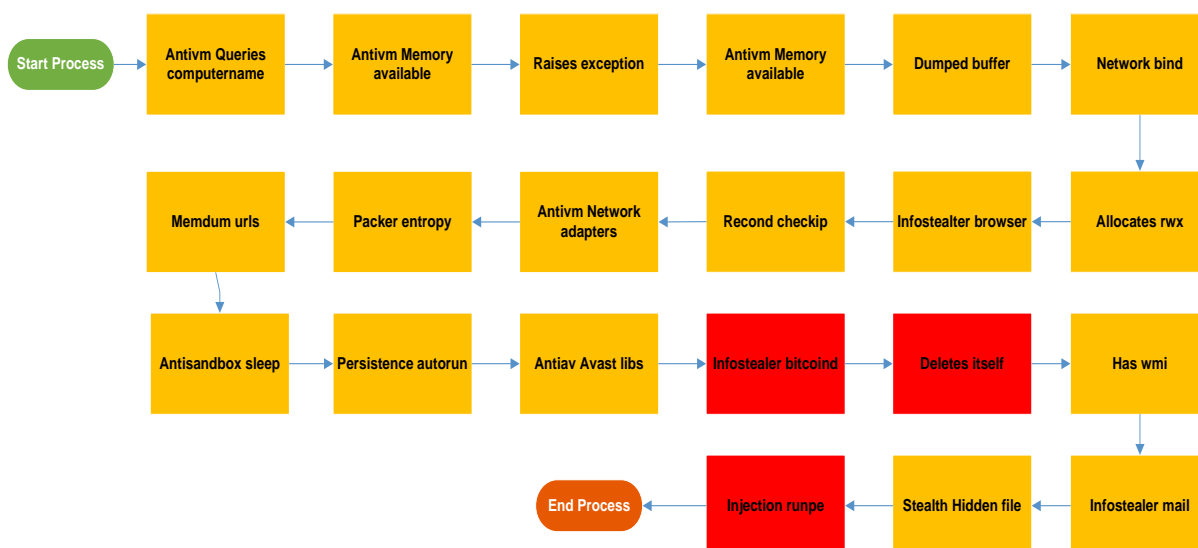


Figure 92: Results Overview

#### 4.2.2.3 Virus Total Results

This is the last step of the analyses process to confirm if the malware artifact is in fact a malware artifact. This process was achieved by running the malware artifact through the online Antivirus scan engine that connects to a maximum of 57 well known antivirus engines. The results from virus total indicated that the artifact that was analysed was classified as a Trojan malware. Only 40 out of 55 antivirus engines were able to determine and confirm that the artifact was indeed a malware file. Figure 93 below provides us with the results from Virus Total and also provided the name of the malware artifact that was analysed further, using static and dynamic analysis techniques as mentioned above.



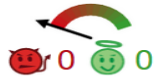
SHA256:	f9460df7c69640e717c32b9aae57473e4a17d8e049df23d3392034cb039fb6e7	
File name:	1123211-090SD.exe	
Detection ratio:	40 / 57	
Analysis date:	2016-09-21 06:31:17 UTC ( 2 months, 3 weeks ago )	
<a href="#">Analysis</a>   <a href="#">File detail</a>   <a href="#">Relationships</a>   <a href="#">Additional information</a>   <a href="#">Comments</a> (0)   <a href="#">Votes</a>   <a href="#">Behavioural information</a>		
Antivirus	Result	Update
ALYac	Trojan.GenericKD.3529436	20160921
AVG	MSIL10.BKWW	20160920
AVware	Trojan.Win32.Generic!BT	20160921
Ad-Aware	Trojan.GenericKD.3529436	20160921
AhnLab-V3	Trojan/Win32.ZBot.N2105680816	20160921
Antiy-AVL	Trojan[HEUR]/Win32.AGeneric	20160921
Arcabit	Trojan.Generic.D35DADC	20160920

Figure 93: Virus Total Results – 1123211-090SD.exe

### 4.3 Chapter summary

In this chapter, the malware analysis process was outlined that was then applied to the malware analysis experiments on the acquired live malware samples. The experiments were carried out in an isolated analysis lab described in chapter 2 (Literature review). Static and Dynamic analysis experiments, which are the two techniques of Malware analysis, were conducted on two live malware samples namely, Ticket\_354041 and 1123211-090SD.exe. These malware samples were two different file formats of which one was in a “Microsoft Word” file and the other a “Portable executable” file that also yielded two different sets of results.

Static analysis allowed us to analyse the malware artifacts through various tools installed on the REMnux and Windows 7, while Dynamic analysis allowed us to safely execute the artifacts in a sandbox environment where the behaviour of the two samples was observed and the results recorded. The Static and Dynamic analysis environments and the tools used are described in detail in chapter 2. For each artifact that was analysed in this chapter, an overview of the results is provided after each section of the analysis experiments.

Based on the preliminary results it was concluded that the samples that were analysed were true live malware samples that were malicious and if they were analysed or executed in a live production environment, we could risk infecting other systems on the network.

The artifacts were also uploaded to an online search system that connects to various antivirus vendors in order to scan, classify and determine severity of the artifact, if it can be found on an infected host. By uploading the malware to the online search system, we were able to

validate the results of our experiments and to also validate that we were in fact working with malicious artifacts. The results of the experiments are discussed in detail in chapter 5.

# CHAPTER 5

## EXPERIMENT RESULTS, DISCUSSION AND CONCLUSION

### CHAPTER OVERVIEW

This chapter focuses on the following areas:

- Introduction
- Experiment Discussion
- Effective Method of Malware Analysis
- Conclusion

### 5 INTRODUCTION

In this chapter we will address one of the research questions as described in chapter 1 of this dissertation in order to be able to provide an answer to the question. We will also further discuss the experiment results in detail and provide a conclusion to this dissertation.

**Research Question:** How to measure the most effective technique of malware analysis and detection on enterprise systems?

#### 5.1 Experiment Results Discussion

In chapter 4 experiments were carried out on the two malware artifacts that we acquired from an online email filtering system. The experiments were conducted using the two methods of malware analysis that are Static and Dynamic analysis methods. Results from these two experiments yielded interesting results that were further validated by uploading our malware artifact onto an online system called Virus total.

##### 5.1.1 Static Analysis Results Overview

###### Malware Files or Artifacts:

- ticket\_354041.doc File
- 1123211-090SD.exe File

The experiment was carried out on the abovementioned file in two environments viz. the REMnux and Windows 7 virtual machines that were setup as part of this research. The experiment started with the static analysis whereby the static or code properties of the file were extracted and studied. The Static analysis experiment was then followed by Dynamic

analysis that then allowed the file to be executed in an isolated environment and its behaviour was observed.

#### 5.1.1.1 Experiment 1 - ticket\_354041.doc File

- The Static analysis process started off by first establishing the hash values of the file in order to uniquely identify the file that the experiment was being conducted on. The hash values that were established are as follows:

```
MD5 = 543c0cf636bc0e56007e6211cd05ecf2
SHA1 = 400cb9f479fd5ab09aa895245e16ba999ce5142e
SHA256 =
3ea894203c48d37b73ce9202dec7eedbf1c724b707f7de058e42c18c3e55bd49
```

- We then needed to confirm the format of the file that we were working on for us to be able to determine the tools that we will use to be able to determine the static properties of the file. The tool used to find the format of the file allowed us to conclude that the file that we were working on was a file created in using the Rich Text format that can also be a Microsoft Word, as indicated from the following results:

```
ticket_354041.doc: Composite Document File V2 Document, Little Endian,
Os: Windows, Version 6.1,
Code page: 1251,
Title: ,
Author: Laura,
Template: Normal.dot,
Last Saved By: Windows, Revision Number: 1,
Name of Creating Application: Microsoft Office Word,
Total Editing Time: 01:00,
Create Time/Date: Wed Oct 19 15:33:00 2016,
Last Saved Time/Date: Wed Oct 19 15:34:00 2016,
Number of Pages: 1, Number of Words: 0,
Number of Characters: 2, Security: 0
remnux@remnux:~/Desktop/Malware/07 November 2016/Extracted/Word Samples$
```

- The file was then ran against the open source antivirus engine installed on the REMnux system to be able to find the signatures of the file and to also see if the antivirus engine can determine if the file was infected. The results from the clam antivirus engine determined that the file was clean and that it was not a malicious document
- We then ran the file against the PEstudio application installed on the Windows 7 system to find IOC that could be obfuscated in the file. The results from the application indicated that the file consisted of 3 out of the 4 IOCs of which these were the following:
  - The files that was referencing Microsoft Office

- The file contained 29 blacklisted strings
  - The file referenced a URL (<http://ns.adobe.com>)
- At this stage we were now able to start suspecting that the file could most probably be a malicious file but we then needed to dive deep into detail from the results produced by the PEStudio application
- The next step in further analysing the file, we needed to extract the black listed strings as earlier indicated. From the extracted strings we were able to find functions or DLLs, URLs, executables and VBA macros that were packed within the file. The following is a summary of the packed content found from the strings and this is not a complete list of results:
  - Functions
    - RtMoveMemory
    - HeapCreate
    - GetModuleHandle
    - RemoveDirectory
  - URLs
    - <http://ns.adobe.com>
  - Executables
    - Microsoft Office Word
  - VBA Macros
    - ThisDocument.cls
    - VBA\_Project
    - Uninstructed.bas
    - Bebop.frm
- We then used another tool to further analyse the obfuscated and this tool was able to extract exact content that is obfuscated that is split into the following areas or categories:
  - Embedded URL
  - Email Address
  - Portable Executables
  - Executable Commands
  - Interesting words that seemed to suggest that they are user accounts
  - OLE Headers
  - Embedded Macros

- The individual VBA macros were then extracted and when the codes were analysed, we found that the code calls three Windows Libraries and one function. The libraries and functions are explained in detail in appendix A.
  - Windows Libraries
    - Kernel32
    - User32
    - ntdll
  - Function
    - RtlAllocateHeap

From the overall results above we could conclude that the file will run using the Microsoft Word application to perform malicious activities. The application will then inject code into another process and run code from the embedded DLL files. This file also consisted of Hex-encoded and Base64-encoded strings that could be used to obfuscate strings.

#### 5.1.1.2 Experiment 2 - 1123211-090SD.exe File

In this experiment the same process that was done on experiment one was also carried out in the same manner for this experiment.

- The Static analysis process started off by first establishing the hash values of the file in order to uniquely identify the file that the experiment was being conducted on. The hash values that were established are as follows:

MD5 = 17a9f0b4bf1cfdd20a492da6620353aa

SHA1 = baf030ac03afef1a0deb3f2b01d4056153732aed

SHA256 = f9460df7c69640e717c32b9aae57473e4a17d8e049df23d3392034cb039fb6e7

- We then needed confirm the format of the file that we were working on for us to be able to determine the tools that we will use to be able to determine the static properties of the file. The tool used to find the format of the file allowed us to conclude that the file that we were working on was a portable executable file as indicated from the following results:

➤ *1123211-090SD.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows*

- The file was then run against the open source antivirus engine installed on the REMnux system to be able to find the signatures of the file and to also see if the antivirus engine can determine if the file was infected. The results from the clam antivirus engine determined that the file was clean and that it was not a malicious document

- We then ran the file against the *PEStudio* application installed on the Windows 7 system to find IOC that could be obfuscated in the file. The results from the application indicated that the file consisted of 1 out of 4 IOCs and these IOCs were the 15 blacklisted strings
- We then investigated the blacklisted strings listed earlier on against two tools that were *PEStudio* and *pestr* respectively. The two tools yielded similar results but the *pestr* yielded more results than the *PEStudio* tool. The *pestr* tools yielded several Windows functions that the malware would use when executed and only 7 of these functions are provided or discussed here. These functions as observed include the following:

- Get\_Computer
- Get\_Application
- Get\_User
- Get\_CurrentDomain
- GetObject
- Create\_Instance
- Dispose\_Instance

These functions indicate that the malware will attempt to get the name of the computer that it is running on, execute one or more applications, find the user that is currently logged on the affected system and also check if the affected system is part of a domain. The malware will then try to get one or more objects, create further instances and then try to delete those created instances in order to hide its tracks

- We then suspected that the executable file contained packed or obfuscated functions. We extracted and decoded suspicious patterns from malicious file using the "*balbuzard.py*" tool. The tool yielded results that suggested that the file contained three executable files that could mean that these files could be either be dropped or executed on an infected system. The executable files extracted are listed below:

- User32.dll - Important Windows Function
- 090SD.exe - Executable
- Mscoree.dll - Important Windows Function

- Apart from the executable files found from the earlier findings we also discovered the following packer within the file:

- Packer            Microsoft Visual C# / Basic .NET

- Packer            Microsoft Visual Studio .NET
- Packer            .NET executable
- Packer            Microsoft Visual C# v7.0 / Basic .NET

- Another discovery from the properties of the file was the .text section that indicated a high Entropy value of 7.72. Entropy is used in different ways by malware authors and analysts, that provides a rough estimation of whether the file is encrypted or not and it helps to determine if further analysis is required

From the overall results above we could conclude that the executable file will run when it is executed within the targeted or infected system and perform malicious activities. The executable then uses the packer or compilers to execute additional Windows functions (“*user32.dll*”, “*mscorlib.dll*”, “*kernel32.dll*”) and also drops another executable file “*090SD.exe*” packed or obfuscated within the executable. The executable also tries to either create a user account on the targeted system called “*POP*”.

### 5.1.2 Dynamic Analysis Results Overview

#### Malware Files or Artifacts:

- ticket\_354041.doc File
- 1123211-090SD.exe File

The experiment was carried out on the abovementioned file on Cuckoo sandbox environment that was set up as part of this research. The experiment followed the static analysis experiments whereby the static or code properties of the file were extracted and studied. In this area Dynamic analysis was performed that allowed the files to be executed in an isolated environment and its behaviour was observed.

From the behaviour of the files when they were executed in the sandbox environment, several behaviour patterns were discovered that were not detected from statically analysing the files. A summary of the results from the Cuckoo analysis is discussed in this section and the full reports are made available as indicated in Appendix E and F.

#### 5.1.2.1 Experiment 1 - ticket\_354041.doc File

The behaviour of the malware file when executed yielded the highlighted results below:

- The malware file first started off by querying for the computer name of the infected system, it then checked the amount of memory in system, and this could be used to detect virtual machines that have a low amount of memory available



- The malware then crashed one or more processes and one or more potentially interesting buffers were extracted; these generally contained injected code, configuration data, etc.
- The malware then started servers listening process and allocated read-write-execute memory (usually to unpack itself)
- Private information was then stolen from local internet browsers and the malware attempted to look up the external IP address that it can connect to from the infected system
- The malware also checked adapter's addresses that were used to detect virtual network interfaces and the binary that was detected was found to be likely containing encrypted or compressed data
- Potentially malicious URLs were found in the process memory dump; one of the processes attempted to delay the analysis task and also installed itself for auto run at Windows start-up
- This malware ended up trying to detect if an antivirus called Avast was installed through the presence of a library
- An attempt to access Bitcoin / ALTCoin wallets was also initiated and the malware then tried to hide itself by deleting its original binary from disk and executed one or more WMI queries
- Information related to installed instant messenger clients was also harvested and attempted to modify windows explorer settings to prevent hidden files from being displayed
- The malware completed its attack on the victim system by executing a process and injected code into it, probably while unpacking

#### **5.1.2.2 Experiment 2 - 1123211-090SD.exe File**

The behaviour of the malware file when executed yielded the highlighted results below:

- The malware crashed one or more processes and one or more potentially interesting buffers were extracted; these generally inject code, configuration data, etc.
- The malware then allocated read-write-execute memory and usually unpacks itself and creates executable files on the file system
- Potentially malicious URLs were found in the process memory dump while the malware was running and the malware then deleted its original binary from disk
- The malware went further by dropping a binary and executed it. It completed its process by executing a process and injected code into it, probably while unpacking

The results from the two dynamic analysis experiments yielded similar behaviour results that could be an indication that the malware files belong to the same category of malware even though the malware files were created in different formats. As previously discussed in chapter

2 on types of malware and then briefly described in the malware lifecycle model, we can make a conclusion that the malware files that were analysed had similar characteristics and behaviour patterns to the Trojan files.

## 5.2 Measurement of Malware Analysis Techniques and detection

This section will assist us to be able to address the research question mentioned in the introduction of this chapter. Before the research question can be addressed, it is important to first recap the two techniques of performing malware analysis already discussed in the Literature review in chapter 3. The conclusion of an effective method of performing malware analysis is based on the experiment results already discussed.

The two techniques of malware analysis are static and dynamic analysis:

- **Static Analysis**

Static analysis of malware is defined as the process of extracting information from malware while it is not running by analysing the code of the malware to determine its true intention (Elisan, 2015).

- **Dynamic Analysis**

Dynamic analysis is defined as the process of extracting information from malware when it is executed (Elisan, 2015). This process entails executing the malware artifact in a secure isolated environment, unlike the static analysis that provides only a view of the malware that is being analyzed.

### 5.2.1 Analysis Method Results comparison

Table 12: Analysis Methods results comparison

Criteria objective	Static Analysis	Dynamic Analysis
<b>Experiment 1 - ticket_354041.doc</b>		
Was the analysis technique able to perform the following:		
1. To determine the hash values of the file.	Yes	Yes
2. To determine the format of the file.	Yes	Yes
3. To detect the file signature of the file that was being analysed.	No	Yes
4. To determine the strings and extract useful information from the strings such as VBA Macros, IP, URLs.	Yes	Yes

<b>Criteria objective</b>	<b>Static Analysis</b>	<b>Dynamic Analysis</b>
5. To determine if the file is packed or included obfuscated code.	Yes	Yes
6. To allow the investigation of the source code to take place.	Yes	No
7. To determine if there was obfuscated code embedded within the source code	Yes	No
8. To determine the characteristics of the malware when executed or using static properties.	Partially	Yes
9. To determine activities that can be done by the malware on the victim system.	Partially	Yes
10. To determine if the malware injects code.	Yes	Yes
11. To determine if the malware attempts to steal user information or modify the system.	No	Yes
12. To determine if the malware drops files.	Yes	Yes
13. To determine if the malware executes any additional process.	No	Yes
14. To determine if the malware deletes itself after it has executed in order to hide itself from being detected.	No	Yes
15. To determine the true intentions and behaviour of the malware	No	Yes
<b>Experiment 2 - 1123211-090SD.exe</b>		
Was the analysis technique able to perform the following:		
1. To determine the hash values of the file.	Yes	Yes
2. To determine the format of the file.	Yes	Yes
3. To detect the file signature of the file that was being analysed.	Yes	Yes
4. To determine the strings and extract useful information from the strings such as VBA Macros, IP, URLs.	Yes	Yes

<b>Criteria objective</b>	<b>Static Analysis</b>	<b>Dynamic Analysis</b>
5. To determine if the file is packed or included obfuscated code.	Yes	Yes
6. To allow the investigation of the source code to take place	Yes	No
7. To determine if there was obfuscated code embedded within the source code	Yes	No
8. To determine the characteristics of the malware when executed or using static properties.	Partially	Yes
9. To determine activities that can be done by the malware on the victim system.	Partially	Yes
10. To determine if the malware injects code.	Partially	Yes
11. To determine if the malware attempts to steal user information or modify the system.	No	Yes
12. To determine if the malware drops files.	Yes	Yes
13. To determine if the malware executes any additional process.	Yes	Yes
14. To determine if the malware deletes itself after it has executed in order to hide itself from being detected.	No	Yes
16. To determine the true intentions and behaviour of the malware	No	Yes

### 5.2.2 Effective Method Criteria Calculation

The two tables below are calculated based on the following classification from the results in table 12 above.

**Yes = 1**

**Partially = 0,5**

**No = 0**

Table 13: Experiment 2 Results

a) Calculations based on experiment 1 results

Experiment 1 - ticket_354041.doc		
Criteria Objective	Static Analysis	Dynamic Analysis
Object 1	1	1
Object 2	1	1
Object 3	0	0,5
Object 4	1	1
Object 5	1	1
Object 6	1	0
Object 7	1	0
Object 8	0,5	1
Object 9	0,5	1
Object 10	1	1
Object 11	0	1
Object 12	1	1
Object 13	0	1
Object 14	0	1
Object 15	0	1
<b>Criteria Total Count</b>	9	12,5

b) Calculations based on experiment 2 results

Experiment 2 - 1123211-090SD.exe		
Criteria Objective	Static Analysis	Dynamic Analysis
Object 1	1	1
Object 2	1	1
Object 3	1	1
Object 4	1	1
Object 5	1	1
Object 6	1	0
Object 7	1	0
Object 8	0,5	1
Object 9	0,5	1
Object 10	0,5	1
Object 11	0	1
Object 12	1	1
Object 13	1	1
Object 14	0	1
Object 15	0	1
<b>Criteria Total Count</b>	10,5	13

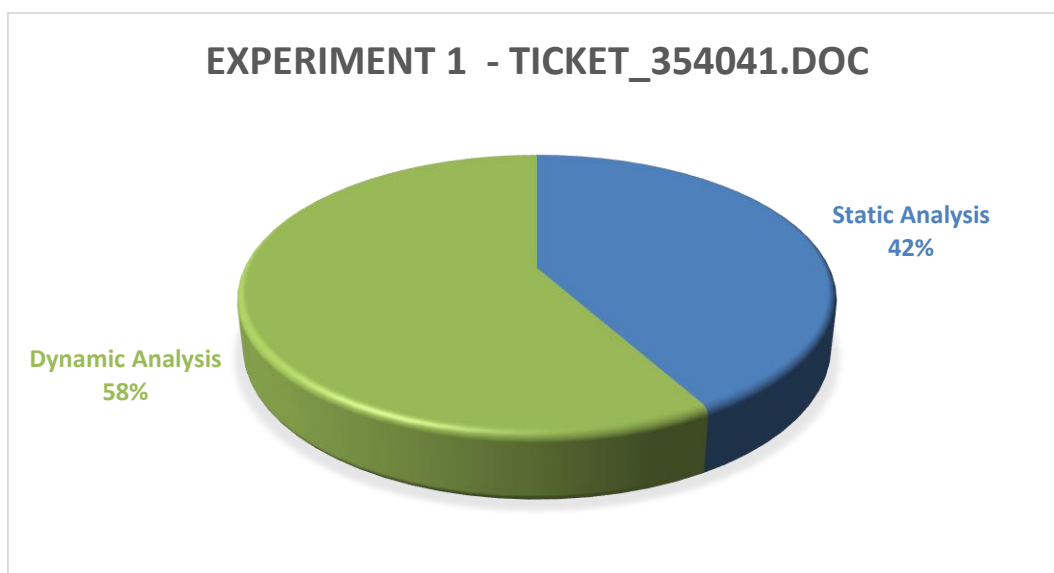


Figure 94: Experiment 1

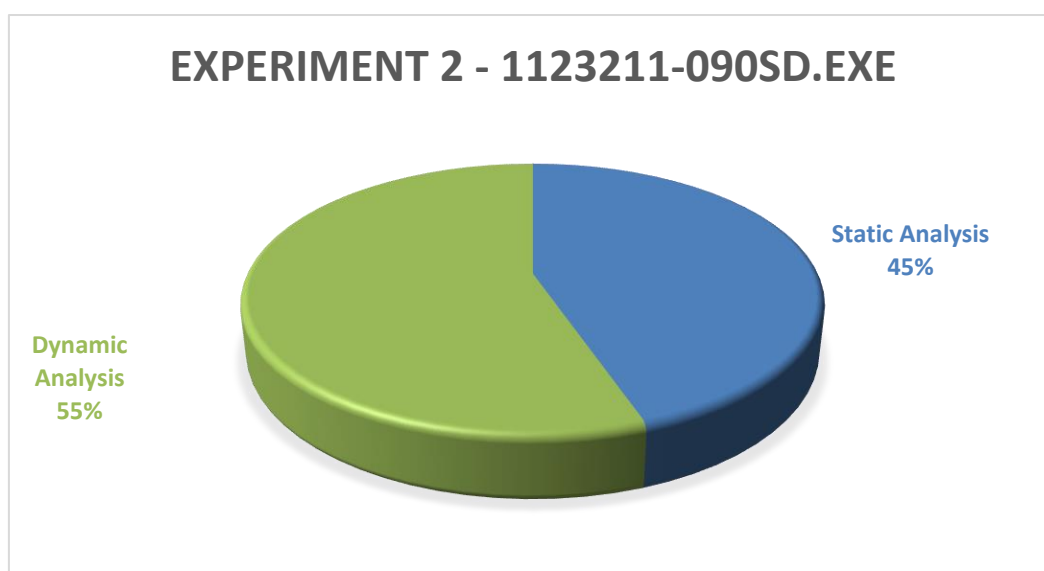


Figure 95: Experiment 2

Based on the above results from section 7.2.1 and 7.2.2 on the comparative results of the two techniques, it is evident that the Dynamic analysis technique of malware analysis was able to address all of the criteria objectives from the described in table 6. From the results we could conclude that Dynamic analysis, if deployed, could be an effective method of performing malware analysis as it provides more information about the true intentions of the malware.

### 5.3 Research Summary

The main research question was to determine how malware detection can be done through malware analysis on enterprise systems.

The main question was broken into four sub-questions and the objective of this research project was to study the important aspects of malware analysis in literature and to determine the most effective techniques to perform malware analysis and detection.

In chapter 1,

- We provided an introduction to malware and malware analysis
- This chapter provided a blueprint on which this research is based as the following areas were discussed: Problem Statement, Research Question, Objectives of Research, Aim of Research and Significance of Study

In chapter 2,

- The literature review of malware and malware analysis was discussed focusing on the theory and background to research of malware, malware analysis, the goals, different techniques and limitation of malware analysis
- The chapter also dove deeply into the malware detection techniques, functionality of malware by focusing on areas such as the behaviour of malware, covert malware launching techniques and malware anti-reverse-engineering techniques that focused on code obfuscation, packers, VBA macros and portable executables
- We discussed and described the setup and configuration of an isolated malware analysis environment that allowed malware samples in chapter 4 to be executed and analyzed in a safe environment. The various tools required for conducting malware analysis experiments were also discussed in this chapter
- Finally, this chapter addressed the three research questions namely:
  - What are the malware analysis and detection techniques that are available in literature?
  - What are the tools and techniques available to effectively perform analysis and detection?
  - How to set up a malware analysis and detection environment?

In chapter 3:

- The study justified the use of quantitative research methodology, and live data samples or artifacts were acquired from an online email filtering system
- The research methodology described in this chapter was the methodology used throughout this dissertation

In chapter 4:

- We conducted practical experiments on live malware samples, provided results and investigations done by the author of this dissertation on the experiments

In chapter 5:

- We discussed and reviewed the results presented from the experiments done in chapter 4 relative to existing research and knowledge from the literature review chapter (chapter 2)

The sub questions of the main research question were address in each chapter of this dissertation, the literature review to the experiments conducted on live samples and results provided.

## **5.4 Research Conclusion and Recommendation**

There is no simple solution to the difficult problems caused by malware. The insecure environment known as the internet and its distributed nature, and important factors such as collaboration and communication also present challenges for securing inter connection computers systems and their networks. Malware has the potential to badly affect any or all users on the internet, from private enterprises to government institutions. While malware often spreads through the internet or no-internet connected systems, it is crucial to remember that malware is malicious software that can be introduced into any computer system in the organization. Using malware directly or indirectly to perform malicious activities online removes trust and assurance from the online community.

In this dissertation, we attempted to address the main objective of this study which was the research problem and the main question of this research, following with its sub questions. We needed to perform malware analysis on malware samples to capture, analyze and record its intrinsic malicious behaviour. This was the main focus in order to be able to establish effective malware detection and analysis techniques.

### **Main Research question**

*“How to effectively perform Malware analysis and detection on enterprise systems in order to reduce the damage of malware attacks on the operation of organizations?”*

The study justified the use quantitative research methodology, with the use of the observation data collection technique or strategy. The data that was collected from the email filtering system include live malware samples. Quantitative data analysis methods were applied to the



data collected. Empirical analysis methods were performed on the quantitative data collected using the simple random method of sampling of data.

An isolated environment was set up for the process of analyzing malware in a safe environment. This dissertation researched and analyzed two different malware samples that were extensively discussed in section 5.1 of chapter 5. We were able to establish that by using malware analysis to analyze the static and dynamic properties of malware, detection capabilities could be put in place to be able to detect and remediate malware infections on enterprise systems. We also learned from the experiments conducted in chapter 4 and from the results, that by using Static and Dynamic analyses techniques of malware analysis, we were able to analyze the properties and behaviour of the malware samples. The two figures below, figures 96 and 97, provide us with a view of the results yielded from the experiments using the two techniques of malware analysis.

Figure 96 shows that:

- Static analysis provided 42% of the results
- Dynamic analysis provided 58 % of the results

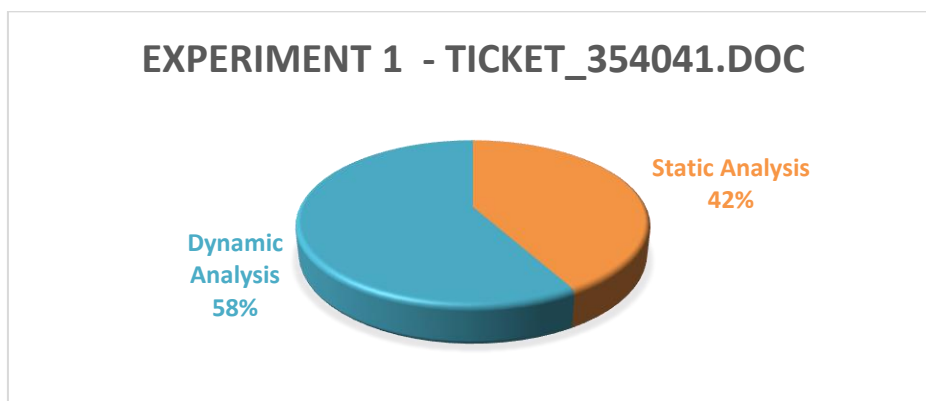


Figure 96: Experiment 1 Analysis and Conclusion

Figure 97 shows that:

- Static analysis provided 45% of the results
- Dynamic analysis provided 55 % of the results

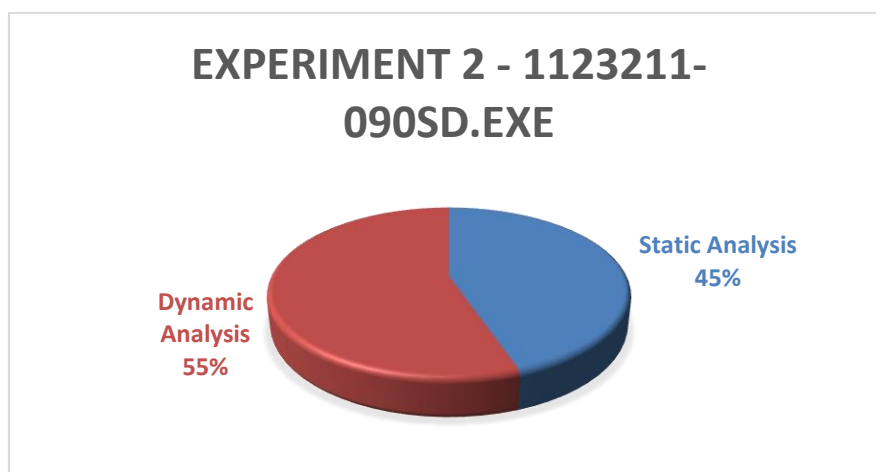


Figure 97: Experiment 1 Analysis and Conclusion.

The two figures show that Dynamic analysis yielded more results than Static analysis.

These values show that we can detect known and unknown malware or malicious software by using a behaviour based strategy which is also known as Dynamic analysis, unlike the use of a manual based strategy in which the attacker can avoid detection by using encryption or obfuscating, but causing the same malicious behaviour without detection.

We have three recommendations and a conclusion:

**Recommendation 1:**

Use Dynamic analysis if the analyst or researcher requires analysing the behaviour of the executed malicious file. This is determined by:

- If the analyst or organisation does not have the required skills to analyse the source code of the malicious file
- If the analyst or organisation has to address or respond to a security breach or malware attack incident that immediate results of the infection are required to be able to detect the malware infection to allow implementation of remedial actions

**Recommendation 2:**

Use Static analysis if the analyst or researcher requires analysing the source code of the malicious file. This will be determined by:

- If the analyst or organisation does have the skills in place to manually analyse malicious file

- If there is no immediate action required such as addressing a security breach or malware incident, and the organisation is only performing static analysis in order to improve their detection capabilities and security countermeasures against malware attacks

### **Recommendation 3:**

Use both Static and Dynamic techniques that will result in a technique known as hybrid analysis. This is determined by:

- If the source code and behaviour of the malicious file need to be determined in order to be able to effectively respond to a security breach or malware incident to be able to implement appropriate detection capabilities of malware

In conclusion, we believe that based on our experiments' results we have contributed to the malware analysis field by adding value to the capabilities of detecting additional attacks of malware on enterprise systems through the malware analyses process.

## **5.5 Future Research**

Malware analysis was done on two different types of malicious file samples and the results yielded similarities in terms of the characteristics and behaviour of the two files. When the files were then uploaded to Virus Total to determine the classification of the malware samples, the results showed that the file samples belonged to the Trojan horse family of malware.

Based on these results our future work will focus more on studying, understanding and analysing further the Trojan malware's characteristics and impact on infected systems. The proposed topic of this study will be "Hybrid Analysis of the Trojan Malware".

# REFERENCES

- BRYMAN, A. & BELL, E. (2007). *Business Research Methods*. New York: OUP Oxford.
- ADAMS, J., KHAN, T.A., RAESIDE, R. & WHITE, D. (2007). *The research methods for graduate business and social science students.*, New Delhi: Sage Publication Ltd.
- AGRAWAL, M., SINGH, H., GOUR, N. & KUMAR, M. A. (2014). Evaluation on Malware Analysis. *International Journal of Computer Science and Information Technologies*. 5. p.3381-3383.
- ALLINGTON, W. (2016). *Tech Support Scams, Fake BSODs, Scareware* . Bits of Technology for Your Library. Available: <http://scls.typepad.com/techbits/2016/02/tech-support-scams-fake-bsods-scareware.html> [Accessed 18 September 2016].
- BHOJANI, N. (2014). Malware Analysis. In: DEFINITION, M. (ed.) *Ethical Hacking*. India: Nirma University.
- BRAND, M.(2010). *Analysis Avoidance Techniques of Malicious Software*. Doctor of Philosophy, Edith Cowan University.
- BURRELL, G. & MORGAN, G. (1979). *Sociological Paradigms and Organizational Analysis*. School of Psychology - Georgia Institute of Technology: Ashgate.
- BURTON, G. (2016). Security. Available: <http://www.computing.co.uk/ctg/news/2443531/total-it-shut-down-at-lincolnshire-county-council-over-zero-day-attack?wb48617274=A77CEB01> [Accessed 27 January 2016].
- CANNELL, J. (2013). Obfuscation: Malware's best friend. *Threat Analysis* . Available from: <https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/> [Accessed 08 March 2016].
- CHALEUNVONG, K. (2009). Data Collection Techniques. Training Course in Reproductive Health Research.
- CHEN, P., HUYGENS, C., DESMET, L. & JOOSEN, W. (2016). Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware. Ku Leuven.
- CHEN, X., ANDERSEN, J., MAO, Z. M., BAILEY, M. & NAZARIO, J. (2008). Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware. IEEE International Conference on Dependable Systems and Networks.
- CHOUDHARY, S., SAROHA, R. & BENIWA, M. S. (2013). How does anti-virus software work? *International Journal of Advanced Research in Computer Science and Software Engineering*. 3. p.483-484.
- CLAUDIO GUARNIERI, A. J. T. & MARK REP SCHLOESSER. (2013). Cuckoo Sandbox. In: CONFERENCE, B. U. (ed.) *Malware Sandbox*. USA: BlackHat USA Conference.

- PERRY, D., PORTER, A. & VOTTA, L. (2000). Empirical studies of software engineering: a roadmap. *Paper presented at the International Conference on Software Engineering Limerick, Ireland.* p.345-355.
- DISTLER, D. & HORNAT, C. (2007). *Malware Analysis: An Introduction.* SANS Institute Reading Room site: Sans Institute.
- DUDOVSKIY, J. (2011). *Research Methodology.* Research-methodology.ne: research-methodology.ne. Available: <http://research-methodology.net/research-philosophy/positivism/> [Accessed 30 January 2017].
- DUGDALE, J. S. (1996). *Entropy and its Physical Meaning. 2nd ed.* Taylor and Francis (UK); CRC (US).
- ELISAN, C. C. (2015). *Advanced Malware Analysis.* United States, McGraw-Hill Education.
- HERBST, F. & COLDWELL, D. (2004). *Business Research.* South Africa: Juta and Co Ltd.
- GRAVETTER, F.J. & FORZANO, LB. (2011). *Research Methods for the Behavioural Sciences.* US - New York, Cengage Learning.
- FOX, W. & BAYAT, M. S. (2007). *A guide to managing research.* South Africa, Juta and Co Ltd.
- GALLAGHER, S. (2014). Risk Assessment / Security & Hacktivism. *BIZ & IT.* Available from: <http://arstechnica.com/security/2014/12/inside-the-wiper-malware-that-brought-sony-pictures-to-its-knees/?wb48617274=36A38774> [Accessed 12 April 2016].
- GOERTZEL, K. M. & WINOGRAD, T. (2009). Tools Report on Anti-Malware. *Types of Malware.* Cyber Security and Information Systems Information Analysis Center Website.
- GUARNIERI, C., TANASI, A. J. & SCHLOESSER, M. R. (2013). Overview of Cuckoo Sandbox. Cuckoo Sandbox.
- GUBA, E. G. & LINCOLN, Y. S. (1994). *Handbook of qualitative research.* US Thousand Oaks, CA: Sage.
- HOWARD, F. (2010). Obfuscation and antiemulation tricks in malicious JavaScript. Available: [https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/malware\\_with\\_your\\_mocha.pdf?la=en.pdf?dl=true](https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/malware_with_your_mocha.pdf?la=en.pdf?dl=true) [Accessed 30 October 2016].
- IDIKA, N. & MATHUR, A. P. (2007). A Survey of Malware Detection Techniques. Purdue University.
- KASAMA, T. (2014). *Study of Malware Analysis Leverating Sandbox Evasive Behaviors.* Degree of Doctor of Philosophy in Engineering, Yokohama National University.
- KASPERSKY. (2016). What is a Trojan Virus. *Internet Security Threats.* Available from: <https://usa.kaspersky.com/internet-security-center/threats/trojans> [Accessed 28 November 2016].
- KENDALL, K. (2007). *Practical Malware Analysis.* Mandiant, *Intelligent Information Security.*

- KERAGALA, D. & WALKER, C. (2016). Detecting Malware and Sandbox Evasion Techniques. [www.sans.org](http://www.sans.org): SANS Institute.
- LANCE. (2013). File Entropy Explained. *File Entropy explained* . Available from: <http://www.forensickb.com/2013/03/file-entropy-explained.html> [Accessed 20 March 2016].
- LIGH, M., ADAIR, S., RICHARD, M. & HARTSTEIN, B. (2010). *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. US. John Wiley & Sons.
- M, S., LEWIS, P. & THORNHILL, A. (2012). *Research Methods for Business Students*. UK. Pearson Education Limited.
- MARAK, V. (2015). *Windows Malware Analysis Essentials*. US. Packt.
- MICHAEL, A., DAVIS, S. M. B. & AARON LEMASTERS. (2010). Hacking Exposed - Malware & Rootkits Secrets and Solutions. *In: EXPOSED*, H. (ed.). US: McGraw Hill.
- MICROSOFT. (2016). *Hyper-V* . Microsoft. Available: [https://technet.microsoft.com/en-us/library/mt169373\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/mt169373(v=ws.11).aspx) [Accessed 12 September 2016].
- MILOVSEVI'C, N. (2013). History of malware. New York: Cornell University.
- MISHRA, U. (2013). Finding and Solving Contradictions of Contradictions of Contradictions of False Positives in Virus Scanning. Cornell University.
- MOHANTY, D. (2017). Anti-Virus Evasion Techniques and Countermeasures. Available: <http://index-of.es/Malware/AVETC.pdf> [Accessed 20 January 2017].
- MORGAN, G. A. & HARMO, R. J. (2001). Data Collection Techniques. *The American Academy of Child and Adolescent Psychiatry*. p.1-4.
- N.BAJPAI. (2011). *Business Research Methods*. Pearson Education India.
- O'LEARY, Z. (2004). *The Essential Guide to Doing Research*. US. Sage Publications.
- OATES, B. J. (2007). *Researching Information Systems and Computing*. US. Sage Publications.
- OSAGHAE, E. O. (2015). Packed Malware Detection using Entropy Related Analysis : A Survey. *IOSR Journal of Engineering (IOSRJEN)*. 5. p.59-61.
- PAGANINI, P. (2015). AV-TEST estimates 12 million new malware variants per month *Malware Statistics* . Available from: <http://securityaffairs.co/wordpress/32352/malware/av-test-statistics-2014.html> [Accessed 23 April 2016].
- PIETREK, M. (1994). Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. Microsoft Website library section: Microsoft.
- PIRSOVCHEANU, R.-S. (2015). *Clustering Analysis of Malware Behavior*. Master Thesis. Aalborg University.

- PRIVACY, O. W. P. O. I. S. A. (2008). *Malicious Software: A security threat to the internet economy*. Organisation for Economic Co-operation and development.
- PURI, R. (2003). *Bots & Botnet: An Overview* Available: <https://www.sans.org/reading-room/whitepapers/malicious/bots-botnet-overview-1299> [Accessed 18 August 2016].
- QUINT, A., LONE-SANG, F., DEDRIE, G., DORSEMAINE, B. & CARLE, D. (2016). *IRMA Sandbox*. Quarkslab. Available: <http://irma.quarkslab.com/index.html> [Accessed 25 September 2016].
- R, V. & RAI, N. (2012). *Windows API based Malware Detection and Framework Analysis*. Sage Journals: International Journal of Scientific & Engineering Research.
- KUMAR, R. (2008). *Research Methodology*. Australia. APH Publishing Corporation.
- REAVIS, J. (2012). *White Paper: The Ongoing Malware Threat*. Available: <https://www.geotrust.com/anti-malware-scan/malware-threat-white-paper.pdf> [Accessed 14 September 2016].
- ROSSOW, C. (2013). *Using Malware Analysis to Evaluate Botnet Resilience*. ter verkrijging van de graad Doctor aan de Vrije Universiteit Amsterdam, Malware Analysis. Vrije Universiteit.
- S, S. Y., PRAYUDI, Y. & RIADI, I. (2015). Implementation of Malware Analysis using Static and Dynamic Analysis Method. *International Journal of Computer Applications*. 117. p.11-15.
- EASTERBROOK, S., SINGER, J., STOREY, M. & DAMIAN, D. (2008). *Guide to Advanced Empirical Software Engineering*. US. Springer.
- SAFFAF, M. N. (2009). *Malware Analysis*. Bachelor of Engineering. Helsinki Metropolia University of Applied Sciences.
- SAIDI, H. (2012). *Challenges in Malware Analysis*. Microsoft.
- SALIHUN, D. (2014). NSA BIOS Backdoor a.k.a. God Mode Malware Part 1: DEITYBOUNCE. *Malware Backdoor Example*. INFOSEC Institute Website: INFOSEC Institute.
- SANABIRA, A. (2007). *Malware Analysis: Environment Design and Architecture*. GCIH Gold Certification Paper. SANS Institute.
- SANS610 (2012). *Reverse Engineering Malware: Malware Analyst Tools and Techniques*. SANS Training.
- SASTRY, D. V. U. K. & KUMAR, K. A. (2012). A Modified Feistel Cipher Involving XOR Operation and Modular Arithmetic Inverse of a Key Matrix. *International Journal of Advanced Computer Science and Applications (IJACSA)*. 3. p.35-39.
- SAVAN GADHIYA & BHAVSAR, K. (2013). Techniques for Malware Analysis. *International Journal of Advanced Research in Computer Science and Software Engineering*. 3. p.972-975.

- SCHIFFMAN, M. (2010). To Hide is to Thrive. *Security*. Available from: [https://blogs.cisco.com/security/a\\_brief\\_history\\_of\\_malware\\_obfuscation\\_part\\_1\\_of\\_2](https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2) [Accessed 23 August 2016].
- SHABAN, F. O. (2013). *Spyware Detection Using Data Mining for Windows Portable Executable Files*. Degree of Master of Science In Information Technology Research. Islamic University of Gaza Deanery of Higher Studies Information Technology program.
- SIKORSKI, M. A. H. (2012). *Practical Malware Analysis*. US. William Pollock.
- SKOUDIS, E. & ZELTSER, L. (2003). *Malware: Fighting Malicious Code*. US. Prentice Hall.
- SVAJCKER, V. (2015). Building A Malware Lab In The Age of Big Data. Available from: <https://community.hpe.com/hpeb/attachments/hpeb/off-by-on-software-security-blog/673/1/Svajcer-VB2015.pdf> [Accessed 20 November 2016].
- SYSTEMS, S. R. T. *Binary XOR Operation*. Available: [http://www.xcprod.com/titan/XCSB-DOC/binary\\_xor.html](http://www.xcprod.com/titan/XCSB-DOC/binary_xor.html) [Accessed 20 October 2016].
- SZOR, P. (2005). *The Art of Computer Virus Research and Defense*, US, Addison-Wesley.
- TAJALIZADEHKHOOB, S. (2013). *Online Banking Fraud Mitigation*. Delft University of Technology.
- TOOL, M. A. (2016). Ole - Forensic Tools. *Tools*. Decalage Website: Decalage.
- TOUCHETTE, F. (2016). The evolution of malware. *Network Security*. p.11-14.
- UPPAL, D., MEHRA, V. & VERMA, V. (2004). Basic survey on Malware Analysis, Tools and Techniques. *International Journal on Computational Sciences & Applications (IJCSA)*. 4. p.10-19.
- VALLI, C. & BRAND, M. (2008). The Malware Analysis Body of Knowledge (MABOK) *The 6th Australian Digital Forensics Conference*. Perth, Western Australia: School of Computer and Information Science, Edith Cowan University, Perth, Western Australia.
- VERGELIS, M., DEMIDOVA, N. & SHCHERBAKOVA, T. (2015). *Spam: features of the quarter*. Securelist. Available: <https://securelist.com/analysis/quarterly-spam-reports/69932/spam-and-phishing-in-the-first-quarter-of-2015/> [Accessed 08 August 2016].
- VIRTUALBOX, S. O. X. (2016). *VirtualBox and Virtualisation Definition*. VirtualBox Website: Oracle. Available: <https://www.virtualbox.org/wiki/Virtualization> [Accessed 25 September 2016].
- VMWARE, I. (2016). *Virtualization Basics - Virtual Machine*. VMWare. Available: <http://www.vmware.com/technology/virtual-machine.html> [Accessed 14 September 2016].
- WALSHAM, G. (2006). Doing interpretive research. *European Journal of Information Systems*. 15. p.20–330.
- WESTCOTT, D. & ZELTSER, L. (2016). REMnux Tools *Tools*. REMnux Website.



WLOSINSKI, L. G. (2015). *The Underground Threat Isaca - Cyber Security 360 Degree Vision*.

ZELTSER, L. (2010). *Reverse Engineering Malware: Tools and Techniques Hands-On*. SANS Institute.

ZELTSER, L. (2015). 5 Steps to Building a Malware Analysis Toolkit Using Free Tools. Available from: <https://zeltser.com/build-malware-analysis-toolkit/> [Accessed 20 November 2016].

ZELTSER, L. (2016). REMnux. *Revers-Engineering and Analyzing Malwre c*. Available from: <https://remnux.org/> [Accessed 25 September 2016].

# **Appendixes Overview**

Appendix A – Important Windows Functions

Appendix B - Common DLLs (Sikorski, 2012)

## **1. Static Analysis Appendixes**

Appendix C - Ticket\_354041

Appendix D - 1123211-090SD.exe

## **2. Dynamic Analysis Appendixes**

Appendix E - Ticket\_354041

Appendix F - 1123211-090SD.exe

Appendix G – Description of CD Contents

## Appendix A

### Important Windows Functions (Sikorski, 2012)

#### 1. **Accept**

Used to listen for incoming connections. This function indicates that the program will listen for incoming connections on a socket.

#### 2. **AdjustTokenPrivileges**

Used to enable or disable specific access privileges. Malware that performs process injection often calls this function to gain additional permissions.

#### 3. **AttachThreadInput**

Attaches the input processing for one thread to another so that the second thread receives input events such as keyboard and mouse events. Key loggers and other spyware use this function.

#### 4. **Bind**

Used to associate a local address to a socket in order to listen for incoming connections.

#### 5. **BitBlt**

Used to copy graphic data from one device to another. Spyware sometimes uses this function to capture screenshots. This function is often added by the compiler as part of library code.

#### 6. **CallNextHookEx**

Used within code that is hooking an event set by SetWindowsHookEx. CallNextHookEx calls the next hook in the chain. Analyze the function calling CallNextHookEx to determine the purpose of a hook set by SetWindowsHookEx.

#### 7. **CertOpenSystemStore**

Used to access the certificates stored on the local system.

#### 8. **CheckRemoteDebuggerPresent**

Checks to see if a specific process (including your own) is being debugged. This function is sometimes used as part of an anti-debugging technique.

#### 9. **CoCreateInstance**

Creates a COM object. COM objects provide a wide variety of functionality. The class identifier (CLSID) will tell you which file contains the code that implements the COM object. See Chapter 7 for an in-depth explanation of COM.

#### 10. **Connect**

Used to connect to a remote socket. Malware often uses low-level functionality to connect to a command-and-control server.

#### 11. **ConnectNamedPipe**

Used to create a server pipe for interprocess communication that will

wait for a client pipe to connect. Backdoors and reverse shells sometimes use `ConnectNamedPipe` to simplify connectivity to a command-and-control server.

## **12. ControlService**

Used to start, stop, modify, or send a signal to a running service. If malware is using its own malicious service, you'll need to analyze the code that implements the service in order to determine the purpose of the call.

## **13. CreateFile**

Creates a new file or opens an existing file.

## **14. CreateFileMapping**

Creates a handle to a file mapping that loads a file into memory and makes it accessible via memory addresses. Launchers, loaders, and injectors use this function to read and modify PE files.

## **15. CreateMutex**

Creates a mutual exclusion object that can be used by malware to ensure that only a single instance of the malware is running on a system at any given time. Malware often uses fixed names for mutexes that can be good host-based indicators to detect additional installations of the malware.

## **16. CreateProcess**

Creates and launches a new process. If malware creates a new process, you will need to analyze the new process as well.

## **17. CreateRemoteThread**

Used to start a thread in a remote process (one other than the calling process). Launchers and stealth malware use `CreateRemoteThread` to inject code into a different process.

## **18. CreateService**

Creates a service that can be started at boot time. Malware uses `CreateService` for persistence, stealth, or to load kernel drivers.

## **19. CreateToolhelp32Snapshot**

Used to create a snapshot of processes, heaps, threads, and modules. Malware often uses this function as part of code that iterates through processes or threads.

## **20. CryptAcquireContext**

Often the first function used by malware to initialize the use of Windows encryption. There are many other functions associated with encryption, most of which start with `Crypt`.

## **21. DeviceIoControl**

Sends a control message from user space to a device driver. `DeviceIoControl` is popular with kernel malware because it is an easy, flexible way to pass information between user space and kernel space.

## **22. DllCanUnloadNow**

An exported function that indicates that the program implements a COM server.

## **23. DllGetClassObject**

An exported function that indicates that the program implements a COM server.

## **24. DllInstall**

An exported function that indicates that the program implements a COM server

## **25. DllRegisterServer**

An exported function that indicates that the program implements a COM server.

## **26. DllUnregisterServer**

An exported function that indicates that the program implements a COM server.

## **27. EnableExecuteProtectionSupport**

An undocumented API function used to modify the Data Execution Protection (DEP) settings of the host, making it more susceptible to attack.

## **28. EnumProcesses**

Used to enumerate through running processes on the system. Malware often enumerates through processes to find a process to inject into.

## **29. EnumProcessModules**

Used to enumerate the loaded modules (executables and DLLs) for a given process. Malware enumerates through modules when doing injection.

## **30. FindFirstFile/FindNextFile**

Used to search through a directory and enumerate the filesystem.

## **31. FindResource**

Used to find a resource in an executable or loaded DLL. Malware sometimes uses resources to store strings, configuration information, or other malicious files. If you see this function used, check for a .rsrc section in the malware's PE header.

## **32. FindWindow**

Searches for an open window on the desktop. Sometimes this function is used as an anti-debugging technique to search for OllyDbg windows.

## **33. FtpPutFile**

A high-level function for uploading a file to a remote FTP server.

## **34. GetAdaptersInfo**

Used to obtain information about the network adapters on the system. Backdoors sometimes call GetAdaptersInfo as part of a survey to gather information about infected machines. In some cases, it's used to gather MAC addresses to check for VMware as part of anti-virtual machine techniques.

### **35. GetAsyncKeyState**

Used to determine whether a particular key is being pressed. Malware sometimes uses this function to implement a keylogger.

### **36. GetDC**

Returns a handle to a device context for a window or the whole screen. Spyware that takes screen captures often uses this function.

### **37. GetForegroundWindow**

Returns a handle to the window currently in the foreground of the desktop. Keyloggers commonly use this function to determine in which window the user is entering his keystrokes.

### **38. Gethostbyname**

Used to perform a DNS lookup on a particular hostname prior to making an IP connection to a remote host. Hostnames that serve as command and-control servers often make good network-based signatures.

### **39. Gethostname**

Retrieves the hostname of the computer. Backdoors sometimes use gethostname as part of a survey of the victim machine.

### **40. GetKeyState**

Used by keyloggers to obtain the status of a particular key on the keyboard.

### **41. GetModuleFilename**

Returns the filename of a module that is loaded in the current process. Malware can use this function to modify or copy files in the currently running process.

### **42. GetModuleHandle**

Used to obtain a handle to an already loaded module. Malware may use GetModuleHandle to locate and modify code in a loaded module or to search for a good location to inject code.

### **43. GetProcAddress**

Retrieves the address of a function in a DLL loaded into memory. Used to import functions from other DLLs in addition to the functions imported in the PE file header.

### **44. GetStartupInfo**

Retrieves a structure containing details about how the current process was configured to run, such as where the standard handles are directed.

### **45. GetSystemDefaultLangId**

Returns the default language settings for the system. This can be used to customize displays and filenames, as part of a survey of an infected victim, or by “patriotic” malware that affects only systems from certain regions.

#### **46. GetTempPath**

Returns the temporary file path. If you see malware call this function, check whether it reads or writes any files in the temporary file path.

#### **47. GetThreadContext**

Returns the context structure of a given thread. The context for a thread stores all the thread information, such as the register values and current state.

#### **48. GetTickCount**

Retrieves the number of milliseconds since bootup. This function is sometimes used to gather timing information as an anti-debugging technique. GetTickCount is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

#### **49. GetVersionEx**

Returns information about which version of Windows is currently running. This can be used as part of a victim survey or to select between different offsets for undocumented structures that have changed between different versions of Windows.

#### **50. GetWindowsDirectory**

Returns the file path to the Windows directory (usually *C:\Windows*). Malware sometimes uses this call to determine into which directory to install additional malicious programs.

#### **51. inet\_addr**

Converts an IP address string like 127.0.0.1 so that it can be used by functions such as connect. The string specified can sometimes be used as a network-based signature.

#### **52. InternetOpen**

Initializes the high-level Internet access functions from WinINet, such as InternetOpenUrl and InternetReadFile. Searching for InternetOpen is a good way to find the start of Internet access functionality. One of the parameters to InternetOpen is the User-Agent, which can sometimes make a good network-based signature.

#### **53. InternetOpenUrl**

Opens a specific URL for a connection using FTP, HTTP, or HTTPS. URLs, if fixed, can often be good network-based signatures.

#### **54. InternetReadFile**

Reads data from a previously opened URL.

#### **55. InternetWriteFile**

Writes data to a previously opened URL.

#### **56. IsDebuggerPresent**

Checks to see if the current process is being debugged, often as part of an anti-debugging technique. This function is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

### **57. IsNTAdmin**

Checks if the user has administrator privileges.

### **58. IsWoW64Process**

Used by a 32-bit process to determine if it is running on a 64-bit operating system.

### **59. LdrLoadDll**

Low-level function to load a DLL into a process, just like LoadLibrary. Normal programs use LoadLibrary, and the presence of this import may indicate a program that is attempting to be stealthy.

### **60. LoadLibrary**

Loads a DLL into a process that may not have been loaded when the program started. Imported by nearly every Win32 program.

### **61. LoadResource**

Loads a resource from a PE file into memory. Malware sometimes uses resources to store strings, configuration information, or other malicious files.

### **62. LsaEnumerateLogonSessions**

Enumerates through logon sessions on the current system, which can be used as part of a credential stealer.

### **63. MapViewOfFile**

Maps a file into memory and makes the contents of the file accessible via memory addresses. Launchers, loaders, and injectors use this function to read and modify PE files. By using MapViewOfFile, the malware can avoid using WriteFile to modify the contents of a file.

### **64. MapVirtualKey**

Translates a virtual-key code into a character value. It is often used by keylogging malware.

### **65. MmGetSystemRoutineAddress**

Similar to GetProcAddress but used by kernel code. This function retrieves the address of a function from another module, but it can only get addresses from *ntoskrnl.exe* and *hal.dll*.

### **66. Module32First/Module32Next**

Used to enumerate through modules loaded into a process. Injectors use this function to determine where to inject code.

### **67. NetScheduleJobAdd**

Submits a request for a program to be run at a specified date and time. Malware can use NetScheduleJobAdd to run a different program. As a malware analyst, you'll need to locate and analyze the program that will be run in the future.

### **68. NetShareEnum**

Used to enumerate network shares.



### **69. NtQueryDirectoryFile**

Returns information about files in a directory. Rootkits commonly hook this function in order to hide files.

### **70. NtQueryInformationProcess**

Returns various information about a specified process. This function is sometimes used as an anti-debugging technique because it can return the same information as CheckRemoteDebuggerPresent.

### **71. NtSetInformationProcess**

Can be used to change the privilege level of a program or to bypass Data Execution Prevention (DEP).

### **72. OleInitialize**

Used to initialize the COM library. Programs that use COM objects must call OleInitialize prior to calling any other COM functions.

### **73. OpenMutex**

Opens a handle to a mutual exclusion object that can be used by malware to ensure that only a single instance of malware is running on a system at any given time. Malware often uses fixed names for mutexes that can be good host-based indicators.

### **74. OpenProcess**

Opens a handle to another process running on the system. This handle can be used to read and write to the other process memory or to inject code into the other process.

### **75. OpenSCManager**

Opens a handle to the service control manager. Any program that installs, modifies, or controls a service must call this function before any other service-manipulation function.

### **76. OutputDebugString**

Outputs a string to a debugger if one is attached. This can be used as an anti-debugging technique.

### **77. PeekNamedPipe**

Used to copy data from a named pipe without removing data from the pipe. This function is popular with reverse shells.

### **78. Process32First/Process32Next**

Used to begin enumerating processes from a previous call to CreateToolhelp32Snapshot. Malware often enumerates through processes to find a process to inject into.

### **79. QueryPerformanceCounter**

Used to retrieve the value of the hardware-based performance counter. This function is sometimes used to gather timing information as part of an anti-debugging technique. It is

often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

#### **80. QueueUserAPC**

Used to execute code for a different thread. Malware sometimes uses QueueUserAPC to inject code into another process.

#### **81. ReadProcessMemory**

Used to read the memory of a remote process.

#### **82. recv**

Receives data from a remote machine. Malware often uses this function to receive data from a remote command-and-control server.

#### **83. RegisterHotKey**

Used to register a handler to be notified anytime a user enters a particular key combination (like CTRL-ALT-J), regardless of which window is active when the user presses the key combination. This function is sometimes used by spyware that remains hidden from the user until the key combination is pressed.

#### **84. RegOpenKey**

Opens a handle to a registry key for reading and editing. Registry keys are sometimes written as a way for software to achieve persistence on a host. The registry also contains a whole host of operating system and application setting information.

#### **85. ResumeThread**

Resumes a previously suspended thread. ResumeThread is used as part of several injection techniques.

#### **86. RtlCreateRegistryKey**

Used to create a registry from kernel-mode code.

#### **87. RtlWriteRegistryValue**

Used to write a value to the registry from kernel-mode code.

#### **88. SamlConnect**

Connects to the Security Account Manager (SAM) in order to make future calls that access credential information. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

#### **89. SamlGetPrivateData**

Queries the private information about a specific user from the Security Account Manager (SAM) database. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

#### **90. SamQueryInformationUse**

Queries information about a specific user in the Security Account Manager (SAM)

database. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

**91. send**

Sends data to a remote machine. Malware often uses this function to send data to a remote command-and-control server.

**92. SetFileTime**

Modifies the creation, access, or last modified time of a file. Malware often uses this function to conceal malicious activity.

**93. SetThreadContext**

Used to modify the context of a given thread. Some injection techniques use SetThreadContext.

**94. SetWindowsHookEx**

Sets a hook function to be called whenever a certain event is called. Commonly used with keyloggers and spyware, this function also provides an easy way to load a DLL into all GUI processes on the system. This function is sometimes added by the compiler.

**95. SfcTerminateWatcherThread**

Used to disable Windows file protection and modify files that otherwise would be protected. SfcFileException can also be used in this capacity.

**96. ShellExecute**

Used to execute another program. If malware creates a new process, you will need to analyze the new process as well.

**97. StartServiceCtrlDispatcher**

Used by a service to connect the main thread of the process to the service control manager. Any process that runs as a service must call this function within 30 seconds of startup. Locating this function in malware tells you that the function should be run as a service.

**98. SuspendThread**

Suspends a thread so that it stops running. Malware will sometimes suspend a thread in order to modify it by performing code injection.

**99. System**

Function to run another program provided by some C runtime libraries. On Windows, this function serves as a wrapper function to CreateProcess.

**100. Thread32First/Thread32Next**

Used to iterate through the threads of a process. Injectors use these functions to find an appropriate thread to inject into.

**101. Toolhelp32ReadProcessMemory**

Used to read the memory of a remote process.

### **102. URLDownloadToFile**

A high-level call to download a file from a web server and save it to disk. This function is popular with downloaders because it implements all the functionality of a downloader in one function call.

### **103. VirtualAllocEx**

A memory-allocation routine that can allocate memory in a remote process. Malware sometimes uses VirtualAllocEx as part of process injection.

### **104. VirtualProtectEx**

Changes the protection on a region of memory. Malware may use this function to change a read-only section of memory to an executable.

### **105. WideCharToMultiByte**

Used to convert a Unicode string into an ASCII string.

### **106. WinExec**

Used to execute another program. If malware creates a new process, you will need to analyze the new process as well.

### **107. WlxLoggedOnSAS (and other Wlx\* functions)**

A function that must be exported by DLLs that will act as authentication module. Malware that exports many Wlx\* functions might be performing Graphical Identification and Authentication (GINA) replacement.

### **108. Wow64DisableWow64FsRedirection**

Disables file redirection that occurs in 32-bit files loaded on a 64-bit system. If a 32-bit application writes to *C:\Windows\System32* after calling this function, then it will write to the real *C:\Windows\System32* instead of being redirected to *C:\Windows\SysWOW64*.

### **109. WriteProcessMemory**

Used to write data to a remote process. Malware uses WriteProcessMemory as part of process injection.

### **110. WSASStartup**

Used to initialize low-level network functionality. Finding calls to WSASStartup can often be an easy way to locate the start of network related functionality.

## Appendix B

### Common DLLs (Sikorski, 2012)

DLL	Description
<i>Kernel32.dll</i>	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
<i>Advapi32.dll</i>	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
<i>User32.dll</i>	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
<i>Gdi32.dll</i>	This DLL contains functions for displaying and manipulating graphics.
<i>Ntdll.dll</i>	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by <i>Kernel32.dll</i> . If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
<i>WSock32.dll</i> and <i>Ws2_32.dll</i>	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
<i>Wininet.dll</i>	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

## Appendix E

Dynamic Analysis - Ticket\_354041

### E.1 Cuckoo Results

**Note: Additional Results available on CD named Folder 1**

## Appendix F

Dynamic Analysis - 1123211-090SD.exe

### F.1 Cuckoo Results -

**Note: Additional Results available on CD named Folder 12**

## Appendix G – CD / USB Contents

Note: The CD/USB provided includes raw data results from experiments conducted using static and dynamic analysis techniques of malware analysis

### CD / USB Contents

#### 1. Full Results from Experiments Conducted

- a) Appendix C
- b) Appendix D
- c) Appendix E
- d) Appendix F

#### 2. Additional Experiments Malware Sample Results

- a) Static Analysis Raw Data Results – Folder Named REMnux Results
- b) Dynamic Analysis Raw Data Results – Folder Named Cuckoo Results