



LETTERKENNY INSTITUTE OF TECHNOLOGY

A thesis submitted in partial fulfilment of
the requirements for the Master of Science in Computing in
Systems & Software Security Letterkenny Institute of Technology

Malware Analysis & Antivirus Signature Creation

Author:

Alan Martin Sweeney

Supervisor:

Mairead Feeney. MSc. BSc

Submitted to Quality and Qualifications Ireland (QQI)
Dearbhú Cáilíochta agus Cáilíochtaí Éireann May 2015

DECLARATION

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in Enterprise Application Development, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of candidate: _____

Date: _____

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of many people at the Letterkenny Institute of Technology and Pramerica Systems Ireland. The author would like to express his sincere gratitude to supervisor Mairead Feeney for her guidance and support. The author would also like to extend his gratitude of the Head of Computing Department Dr Thomas Dowling, and all the staffing Letterkenny Institute of Technology for their help and support of the years. Finally the author would like to thank Anthony Caldwell for his support throughout the writing of this thesis.

ABSTRACT

The rapid advances social media, educational tools and communications platforms available today have expanded the attack landscape through which the malicious user can propagate their work can carry out damaging attacks. Attacks against desktop, mobile and cloud-based systems have seen a sharp increase in recent years owing to recent advanced malware creation techniques and all the more worrying are the common misconceptions among end-users that anti-malware programs will safeguard against these threats. Progressive analysis of these malware specimens has prompted the security industry as a whole to take the matter more seriously but currently, appears to be reacting to threats rather than pro-actively building defences against the next wave of attacks. Significant difficulties are faced by the security industry in this respect. On this basis, the following work evaluates and analyses a Windows malware specimen in a controlled virtual environment to determine its purpose and function using a combination of static and dynamic code analysis. Results show that obfuscation strategies employed by malware writers 'morph' viruses into forms which evade detection even by complex heuristic detection algorithms. It is recommended that the security process including the policies, procedures and security awareness training programmes be actively developed in the corporate context and that end-users in the domestic case take greater care with downloading.

CONTENT

DECLARATION	2
ACKNOWLEDGEMENTS	3
ABSTRACT.....	4
TABLE OF FIGURES	8
1. INTRODUCTION.....	10
1.1 Purpose	10
1.2 Background	10
1.3 Aims & Objectives of Research	11
1.4 Outline of Report	12
2. LITERATURE REVIEW	13
2.1 Introduction	13
2.2 A Brief History of Malware.....	13
2.3 Malware Evolution	14
2.4 Malware Propagation	14
2.5 The Failure of Anti-Malware Systems	15
2.6 Obfuscation Techniques	15
2.6.1 Packing Malware	15
2.6.2 Encrypted Malware.....	15
2.6.3 Oligomorphic Malware	16
2.6.4 Polymorphic Malware	16
2.6.7 Metamorphic Malware	17
2.6.8 Garbage Code Insertion	18
2.6.9 Register Assignment Substitution	19
2.6.10 Code Transposition	19
2.7 Approaches to Malware Analysis.....	20
2.7.1 Static & Dynamic analysis advantages & disadvantages	20
3 DESIGN	22
3.1 Introduction	22
3.2 Fully-Automated Analysis	22
3.3 Static Analysis.....	22
3.4 Dynamic Analysis	23

3.5	Interactive Behaviour Analysis.....	23
3.6	Combining Analysis Stages.....	23
3.7	Signature Creation	24
3.7.1	Hash Signatures.....	24
3.7.2	File Diffing	26
4	IMPLEMENTATION	28
4.1	Lab design, environment configuration and analysis report structure.	28
4.1.1	REMnux Machine Configuration	29
4.1.2	Windows 7 Machine Configuration	29
4.2	Malware Analysis Report Structure	29
4.3	Malicious Specimen Selection.....	30
5	EVUALATION & TESTING	32
5.1	Introduction	32
5.2	Malware Analysis Report	32
5.3	Specimen Selection & Testing.....	32
5.4	Static Analysis.....	34
5.5	Dynamic Analysis	38
5.6	Interactive Behavioural Analysis.....	41
5.7	Malware Mitigation Strategy	41
5.7.1	Signature creation in REMnux.....	41
5.7.2	Signature Transferred to Windows 7 Machine	42
5.7.3	Signature Validation on Windows 7 Machine via Command Line	42
5.7.4	Malware Removal Validation on Windows 7 Machine.....	43
5.8	Summary of Findings.....	44
6	CONCLUSION.....	45
	Limitation of Research	46
	Further Work.....	47
7	REFERENCES	48
8	APPENDICS	52
	Appendix 1: Glossary Of Terms.....	52
	Appendix 2: FireEye Report	52
	Appendix 3: REMnux Machine Configuration.....	55

Step 3	55
Step 4	55
Step 5	55
Step 6	55

TABLE OF FIGURES

Figure 1: Basic encryption and decryption processes.....	14
Figure 2: Polymorphic Malware Comparing “Backdoor.Win32.Shiz.pe” Shiz PE files:.....	17
Figure 3: Metamorphic Trojan w32 NGVCK (Next Generation Virus Creation Kit).....	18
Figure 4: Garbage Code Insertion: operand register (NOP) values insertion	19
Figure 5: Register Assignment Substitution.....	19
Figure 6: Code Transposition: transposition based on Unconditional Branches.....	20
Figure 7: Malware Dissection Analysis Framework	22
Figure 8: A Single Byte Change from “00” To “01” Results In A Unique Hash Signature.....	25
Figure 9: MD 5 Hashed Signature Portions	25
Figure 10: Sectional Hashing of Pandora RAT and Hash Base Signature Creation	25
Figure 11: ClamWin AV String Signature.....	26
Figure 12: Diffing with Wildcard Characters When Section Offsets Are Present	27
Figure 13: Lab Design and Connectivity	28
Figure 14: Windows 7 ClamWin MaliciousSpecimen01.exe scan results	33
Figure 15: REMnux ClamAV MaliciousSpecimen01.exe scan results.....	33
Figure 16: Pyew - PE file embedded signature test	34
Figure 17: Pehash generating the MD5, SHA1 and SSDEEP hash of the malware specimen.	35
Figure 18: VirusTotal analysis Result	35
Figure 19: FireEye Report.....	35
Figure 20: Pescan is used to understand the behaviour of an executable	36
Figure 21: upx unpacker.....	36
Figure 22: pescan after unpacking.....	36
Figure 23: pestr output of a network related string.....	37
Figure 24: Readpe results before unpacking of MaliciousSpecimen01.exe	37
Figure 25: Readpe results after unpacking MaliciousSpecimen01.exe	38
Figure 26: Vivisect Disassembler Imports	38
Figure 27: Vivisect Disassembler Import.....	39
Figure 28: Vivisect Disassembler Function Dependencies.....	39
Figure 29: Vivisect Disassembler Function Dependencies.....	39
Figure 30: Vivisect Disassembler Function Dependencies.....	40
Figure 31: Vivisect Disassembler Function Dependencies.....	40
Figure 32: Internet URL connection	40
Figure 33: Wireshark Capture from REMnux.....	41
Figure 34: MD5 hash based signature for MaliciousSpecimen01.exe	42
Figure 35: MaliciousSpecimen01.hdb signature validation test.....	42
Figure 36: Signature Transferred to Windows 7	42
Figure 37: Signature Validation on Windows 7 Machine.....	43
Figure 38: ClamWin AV Scan.....	43
Figure 39: “MaliciousSpecimen01.exe” moved to quarantine	43

Figure 40: Rescan to validate " MaliciousSpecimen01.exe" removed 44

1. INTRODUCTION

1.1 Purpose

Malicious executables are designed to infect system's files, manipulate and control operating system (OS) by taking advantage of system compatibilities to ensure self-survival from variant to variant. Windows portable executable (PE) file structure remains similar between available versions of the windows OS thus making PE files a target to camouflage malware. This work elaborates upon the failure of modern antivirus software to detect malicious software/malware. The literature review focuses on a short history of malware, its propagation and the failure of current anti-malware tools. The rapid development and propagation of malware has significantly influenced how security practitioners and malware experts have had to respond to these threats. Accordingly, the static and dynamic analysis of malware is discussed as part of the methodology to be employed leading to a hybrid design augmented by analysis techniques. Following its dissection, the removal of the malware involves the disconnection of the infected computer from its network, deletion of its artefacts and the creation of a malware signature.

1.2 Background

Contemporary society has been quick to adopt the rapid advances in monetary instruments, social media, educational tools and communications platforms available today (Forum, 2014). The perceived positive advancements in these areas have been tempered by industrial and academic reports documenting the considerable difficulties faced from a security perspective. Cybercriminals, aiming to expose and exploit cracks in software to carry out malicious attacks can reverse-engineer released patches, check for flaws and in some cases target older, especially unsupported, software (TrendLabs, 2014). Significantly, key trend indicators from industrial reports support the contention that mobile malware, particularly malware targeted at Google's Android operating system is increasing (Kaspersky, 2013; Verizon, 2014), through point-of-sale vendor systems which have been compromised via Zeus malware (Verizon, 2014), and authentication malware such as PERKEL and ZITMO (TrendLabs, 2014) unfortunately, indicating that malware threats will continue.

Malware refers to viruses, worms, ransom-ware, Trojan horses, key-loggers, root-kits, spyware, diallers, malicious Browser Helper Object BHOs (BHOs), adware and malicious programs (Kramer & Bradfield, 2010; Szor, 2005). Recent research indicated that common misconceptions among end-users that anti-malware programs that will safeguard against these threats exists (Schiffman, 2010). Indeed it is this complacency among the end-user that the malware designer relies. Malware designers make progressive augmentations of their products to evade detection (Konstantinou & Wolthusen, 2008; Balakrishnan & Schulze, 2005) and more worryingly a significant escalation in malware strains in 2013 has been reported, averaging

82,000 per day (Panda Security, 2013). Detection and prevention efforts have focused upon signature-based anti-malware products; however these may be circumvented via obfuscation techniques to conceal malicious codes true purpose (Shafiq, et al., 2009).

Featuring prominently in academic research and industrial research, static analysis remains the principal technique for client based malware detection relying on signature based detection methods (Griffin, et al., 2009; Christodorescu & Jha, 2004). Safeguarding against malicious software traditionally relies on signature-based anti-malware programs that utilise a pre-defined set of signatures and heuristic methods (Shafiq, et al., 2009); however, malware authors have adopted obfuscation. Anti-malware programs use syntactical patterns or regular expressions to generate a database of characteristics which identify known malware variants. Obfuscation is the deliberate act by malicious software authors to conceal malicious codes true purpose while retaining the functionality of its original state (Balakrishnan & Schulze, 2005). Obfuscation technology was originally developed to protect the intellectual property of lawful software authors, nevertheless obfuscation techniques have been adopted by malicious software authors to foil detection by anti-malware products (Schiffman, 2010; Konstantinou & Wolthusen, 2008). Obfuscation mechanisms deployed to circumvent signature based detection include encryption, oligomorphism, polymorphism, metamorphism and packing. This is achieved by hiding the malicious functionality within data sections of the binary that give the impression of being different for each variant.

1.3 Aim & Objectives of Research

The research question for this work will focus on conducting, evaluating and analysing a malicious Windows PE file in a controlled virtual environment to determine the purpose and function of the specimen/artefact. Firstly it will be shown that the specimen is undetected by antivirus, online services i.e., VirusTotal and commercial detection application. Static and dynamic code analysis aid in this analysis using REMnux which is a Linux distribution used for malware analysis. The infected OS will be isolated and the necessary monitoring tools will observe the specimen/artefacts execution during the analysis phase to the C2 (Command & Control) server using tools including Wireshark and other analysis tools contained within the REMnux distributions. The resulting analysis will expose the specimen/artefacts exploit and there after create a signature which uniquely identifies that malicious PE file stopping future infections thus illustrating that detection is only possible when a specimen has been previously identified.

1.4 Outline of Report

Chapter 2: Literature review of the evolution & approaches to malware analysis

Chapter 3: Design, detail and examine a proposed malware dissection analysis framework.

Chapter 4: Implementation, requirements specification and design.

Chapter 5: Evaluation and Testing, analyses the results of the testing and discussion on findings.

Chapter 6: Conclusion, limitations and suggestions for further research

2. LITERATURE REVIEW

2.1 Introduction

Given the increasing prevalence of malware and its variants across diverse systems, their efficacy is magnified by the inability of anti-virus products to adapt, thus making removal complicated and a laborious task, FireEye research indicates that 90% of malicious binaries morph within an hour of creation (FireEye, 2014). As advanced malware evolves, organizations have to contend with system downtime, loss or compromise of data, financial loss, brand damage and/or loss of customers, damage to the integrity or delivery of critical goods, services or information, damage to systems and other organizations systems compromised.

2.2 A Brief History of Malware

The physicist John Von Neumann (1903-1957) developed the earliest theoretical perspective on malware/viruses (GDATA, 2014). While purely an interesting theoretical artefact of Von Neumann's automatons (or organisms), the primitive conceptualization of the virus was established (Von Neumann, 1951). By 1986, viruses were most definitely not theoretical and the term 'computer virus', attributed to Fred Cohen (Scientific American, 2001), was becoming part of mainstream computer science thought. By this point, most viruses were only to be found in universities where their propagation excelled via infected floppy disks (Solomon, 1993). Among the most famous were, Brain. A (1986), Lehigh, Stoned, and Jerusalem (1987), the Morris worm (1988) and Michelangelo (1991) all of which caused considerable damage for some time (Dwan, 2000). With the networked computer era, many new vectors by which the virus/malware could propagate permitted the propagation of viruses at an alarming rate. By the mid-90s, businesses were equally impacted mainly due to macro viruses such as Melissa (1999) and Kak (1999). In the new millennium, and with the exponential growth of the Internet, email worms such as Loveletter, the Anna Kournikova email worm, Magistr and CodeRed saw the evolution of the virus into malware into variants which were becoming increasingly dangerous and more difficult to detect. In the contemporary case, the advent of the mobile device has allowed us to interact with and manage our work and personal data on an unprecedented scale. In the post-pc era, mobile devices including tablets, phones, netbooks and laptops present a more attractive, practical and economical alternative to desktops. In 2013, it was predicted that by 2015, there will be in excess of two billion mobile devices (AWPG, 2013), recent reports indicate that it's closer to seven billion (Boren, 2014). By default, these devices interconnect to create small-scale networks so that data from the users' online profiles are synchronized with the data stored on the mobile device. Applications or 'apps' are the prime delivery mechanism for tools, services and connectivity on the mobile device but also malware and viruses. This is of some concern since mobile users are downloading applications with little regard for the consequences of doing so and concordantly, a rise in the number of attacks via the 'app' is becoming increasingly problematic (McAfee, 2014). Among the

malicious behaviours that malware can be attributed to, mobile malware can make calls without the user’s permission, install additional applications without the user’s permission, monitor incoming SMS messages and record them, reveal GPS location to name a few. Unlike the application and programs that run on a standard PC, laptop, tablet or otherwise, malware obscures its origin and intent (Ször & Ferrie, 2001) and problematically, largely goes unrecognized by the operating system since its intent is not revealed until it is executed or unpacked. It is of some interest in the work presented here to describe the mechanisms by which malware has evolved, the characteristics of its propagation and why anti-virus software fails to detect such a threat.

2.3 Malware Evolution

It is the specific purpose of malware to evade detection and when triggered, **execute** the hacker’s objectives for the malware. The approaches deployed by malware developers utilize web, email and file downloads to target the end-user. By design, malware obfuscates itself in order to circumvent signature based detection. This is achieved with techniques such as, packing, encryption, oligomorphic, polymorphism, metamorphism to name a few. Other methods include more sophisticated stealth techniques to hide malicious functionality within data sections of the binary file making each variant significantly different. Figure 1 shows how a basic encryption system works where a clear text message is encrypted to produce cipher text with the reverse being to take cipher text and decrypt to clear text so that it may be read and understood as illustrated in Figure1. This is an important feature of malware evolution since encryption is used to obfuscate malware and evade detection.

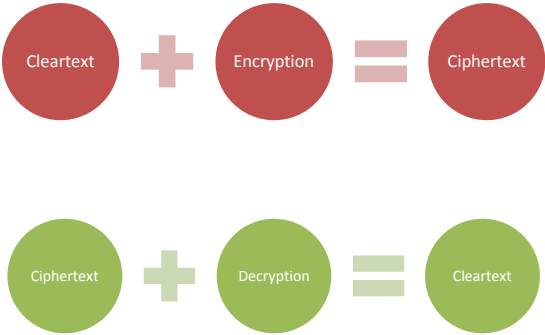


Figure 1: Basic encryption and decryption processes

2.4 Malware Propagation

The modus operandi (operating method) of any installed malware is firstly to establish a firm foundation with the victims systems and once this is achieved, to propagate its code as widely as possible. Generally, this involves setting up a C2 channel. The C2 channel transmits the

identity and location of the victim and typically advances the cause of the malware by downloading additional augmentations to elevate privileges, establish more connections to the host system and otherwise cause further damage. Therefore malware propagating via the C2 channel may have the ability to spread rapidly, infecting more devices, and cause considerable disruption to the victims system without detection.

2.5 The Failure of Anti-Malware Systems

Anti-virus remains a vital component of IT security architecture as organizations utilize it as a cleaning mechanism complemented by advanced security (FireEye, 2014). While security experts agree that any technical solution that enhances the security profile of an organization is welcome much research suggests that signature based detection techniques employed by anti-virus products have weaknesses exploited by the malware designer. These weaknesses include reliance on the signature database that require constant updating from both vendor and consumer, the signature database is a snapshot in time which is immediately out-dated once released to the consumer and self-modifying malware will defeat signature based detection engines. The malware code's true intent is obscured through packing and encrypting (Ször & Ferrie, 2001) since, as far as the operating system is concerned, the codes bona-fide intent is not revealed when unpacking to machine readable code. Although, obfuscation techniques include, but are not limited to, oligomorphic, polymorphic (Song, et al., 2007) and metamorphic (O'Kane, et al., 2011) all of which are designed to decrease the probability of their detection by automated tools, a common approach used by the malware designer is the packing technique which compresses malicious software.

2.6 Obfuscation Techniques

2.6.1 Packing Malware

Packing is a technique used by software companies in the distribution and deployment of their software and as a result, any software must be unpacked for analysis before a piece of malicious software is detected. The compression of malicious software to evade detection by antimalware tools (O'Kane, et al., 2011) maybe detected by entropy analysis (Lyda & Hamrock, 2007) which uses the statistical techniques involving the use of data randomization patterns as indicators of packed or encrypted data sets and therefore might indicate an encrypted executable that otherwise my go unnoticed (Forensickb, 2013).

2.6.2 Encrypted Malware

Cryptography is an approach adopted to evade signature based detection utilizing an encryptor/decryptor, initially observed in 1987 used on the CASCADE virus (Schiffman, 2010; Beaucamps, 2007). Two elements are present in encrypted malware, a decryption loop and main body. The decryption loop, encrypts and decrypts code of the main body which is actual malicious code, encrypted. The decryptor loop is responsible for decoding the main body into

machine executable code and meaningful data prior to the execution on the host computer. Encrypting malicious software can be achieved via one-to-one mapping transforming code byte by byte, zero-operand instruction such as INC (increment) instruction or the NEG (negate) instruction and reversible instruction including ADD or XOR with random keys. All of which is important as detection is not immediate, signatures are employed for detection, and therefore, unknown malicious software can evade detection. However, there is the possibility of indirect detection, if enough bytes are present forming string signature or through the decryption code pattern which remains constant on each subsequent infection a variant of a known specimen may be detected.

2.6.3 Oligomorphic Malware

Discovered in 1990 as a Denial of Service (DoS) attack, Oligomorphic malware engines prolong detection by selecting at random pieces of the decryptor from a number of predefined alternatives evading code pattern signature detection. Malicious software authors devised Oligomorphic malware or Semi-Polymorphic malware, both approaches make the decryptor loop of encrypted malware mutate by substituting the decryptor code in each new infection (Rad, et al., 2011; Szor, 2005). Oligomorphic malware has a collection of decryptor's, randomly chosen for each infection; therefore the decryptor code is noticeably different in multiple occurrences of that given executable.

2.6.4 Polymorphic Malware

The limitation of Oligomorphic malware where overcome by the creation of Polymorphic malware designed to generate an unlimited number of divergent decryptor's and the use of obfuscation methods therefore, making analysis of that malware specimen difficult by changing its appearance. Mark Washburn developed the first know Polymorphic malicious virus, Chameleon which is 1,260 bytes in length (Schiffman, 2010; Wong & Stamp, 2006; Szor, 2005). With the aid of mutation engines such as "The Mutation Engine (MtE)" and obfuscation methods that mutate code including junk/dead code insertion, code transposition, Variable/Register substitution, instruction replacement and instruction permutation (Xufang, et al., 2011; You & Yim, 2010; Xufang; Bruschi, et al., 2007). Polymorphic malwares is designed to obstruct signature detection however the malicious codes body that appears after decryption, can be used as a source for detection through exploitation of emulation technique resulting in simple string matching, (Schiffman, 2010) thus defeating malicious software authors attempts of camouflaging their malwares . Comparing two instances or malware with the same code structure such as "Backdoor.Win32.Shiz.pe" that employees mutation engines and obfuscation methods result in data that is completely different as illustrated in **Figure 2** (Lavasoft, 2014) which is comparing a windows backdoor "Backdoor.Win32.Shiz.pe" Shiz PE files: 0a522256764f748f6c89fc76ddc519f2 (295936 bytes in size) and 04c359648091980d36bdba07149b16f7 (280064 bytes in size)

003F8: 00 00 00 00 00 00 00 00 	003F8: 00 00 00 00 00 00 00 00
00400: FF 35 34 87 40 00 51 FF я54+@.Qя	00400: FF 15 08 B0 47 00 8B C8 я...°G.<И
00408: 15 94 60 40 00 8B CF 8B ."`@.<Пк	00408: 43 68 B4 31 00 00 FF 15 Chr1...я.
00410: CB FF 15 88 61 40 00 03 Ля.Са@..	00410: 7C B0 47 00 21 F5 8B C8 °G.!x<И
00418: EB 8D 05 3A A0 40 00 50 лК.: @.P	00418: 4F 6A 00 6A 00 FF 15 00 Oj.j.я..
00420: 68 57 76 40 00 FF 15 DC hWv@.я.ь	00420: B0 47 00 83 CE 4A 40 FF °G.fOJ@я
00428: 60 40 00 03 DA 2B EF 8B `@..Ъ+n<	00428: 15 34 B0 47 00 81 C0 33 .4°G.íA3
00430: D7 49 FF 15 DC 60 40 00 ЧТя.ь`@.	00430: 00 00 00 6A 00 8D 05 0B ...j.К..
00438: 46 45 FF 15 A0 61 40 00 FEя. а@.	00438: 84 40 00 50 8D 05 80 6B „@.PK.Ъk
00440: 2B F1 03 CF 8B C7 68 F9 +с.П<Эшц	00440: 40 00 50 FF 15 68 B0 47 @.Pя.h°G
00448: E5 26 00 FF 15 00 61 40 е&.я...а@	00448: 00 2B FA 4F 56 FF 15 74 .+°OVя.т
00450: 00 83 CB 1A A1 ED 90 40 .рЛ.Ўнђ@	00450: B0 47 00 03 DE 46 FF 15 °G..ЮPя.
00458: 00 50 68 F8 93 40 00 FF .Phm`@.я	00458: 94 B0 47 00 BF 16 00 00 "°G.i...
00460: 15 D4 61 40 00 83 C8 42 .фа@.рИВ	00460: 00 45 FF 15 7C B0 47 00 .Ея. °G.
00468: 42 FF 0D 51 85 40 00 6A Вя.Q...@.j	00468: 81 EF 1E 00 00 00 47 6A Ѓп....Gj
00470: 00 6A 00 68 BC 7A 40 00 .j.hjz@.	00470: 16 68 B9 A0 40 00 57 FF .hM @.Wя
00478: FF 15 60 60 40 00 46 83 я.`@.Ff	00478: 15 14 B0 47 00 B8 45 00 ...°G.аЕ.
00480: CE 1E 42 8B 1D 54 A2 40 O.В<.Тў@	00480: 00 00 FF 15 70 B0 47 00 ...я.p°G.
00488: 00 53 50 FF 15 70 60 40 .SPя.p`@	00488: 8B F2 4F 8D 05 B0 83 40 <тOK.°р@
00490: 00 03 C1 33 F0 45 68 9B ..E3pEh>	00490: 00 50 FF 15 68 B0 47 00 Pя.h°G.
00498: 7C 40 00 FF 15 8C 60 40 @.я.ь`@	00498: 03 F3 FF 15 7C B0 47 00 .уя. °G.
004A0: 00 2B D8 03 C1 FF 35 F0 .+Ш.Ея5p	004A0: 20 F3 83 25 92 9F 40 00 yр%’ұ@.
004A8: 97 40 00 8D 05 8C 8E 40 -@.К.Бн@	004A8: 7A 8D 05 8A 73 40 00 50 zК.Бс@.P
004B0: 00 50 8B 1D FC 99 40 00 .P<.ь™@.	004B0: FF 05 5D 98 40 00 51 FF я.]@.Qя
004B8: 53 FF 15 00 62 40 00 B9 Sя...b@.№	004B8: 15 3C B0 47 00 2B F3 49 .<°G.+yI
004C0: 14 00 00 00 50 6A 50 8B ...PjP<	004C0: FF 15 08 B0 47 00 BF 64 я...°G.id
004C8: 0D 75 7F 40 00 51 FF 15 .u@.Qя.	004C8: 00 00 00 8B D9 8B 15 F0 ...<Ц<.p
004D0: 08 62 40 00 03 F3 33 EE .b@...y3o	004D0: 70 40 00 52 6A 6B 52 FF p@.RjkPя
004D8: 50 68 12 9E 40 00 8D 05 Ph.h@.К.	004D8: 15 1C B0 47 00 03 C6 03 ...°G..Ж.
004E0: BD 85 40 00 50 FF 15 BC S...@.Pя.j	004E0: CA 4F 8D 05 1C 98 40 00 KOK...@.
004E8: 60 40 00 83 CE 50 47 8D `@.fOPGК	004E8: 50 8D 05 02 62 40 00 50 PK...b@.P
004F0: 05 27 7F 40 00 50 FF 15 .'@@.Pя.	004F0: 68 15 6D 40 00 FF 15 68 h.m@.я.h
004F8: 08 61 40 00 33 CE 6A 00 .a@.3Oj.	004F8: B0 47 00 03 D8 FF 15 70 °G..Шя.p
00500: FF 15 2C 60 40 00 B9 3A я.,`@.№:	00500: B0 47 00 41 83 C9 16 FF °G.AрЙ.я
00508: 00 00 00 48 8D 05 A9 94 ...HK.@”	00508: 15 84 B0 47 00 2B D0 40 ...°G.+P@
00510: 40 00 50 8B 3D 7C 97 40 @.P<= -@	00510: 8D 05 AB 96 40 00 50 83 К.«-@.Pф
00518: 00 57 FF 15 AC 61 40 00 .Wя.¬а@.	00518: 25 03 97 40 00 36 68 10 %. -@.6h.
00520: 8B FA 03 DA 4E FF 15 E0 <ъ.ЪНя.а	00520: 75 29 00 FF 15 58 B0 47 u).я.X°G
00528: 60 40 00 83 CA 30 87 EB `@.рKO+л	00528: 00 48 66 BF CC 4A FF 15 .HfIMJя.
00530: FF 15 CC 61 40 00 8D 3D я.Ma@.К=	00530: 78 B0 47 00 2B C8 6A 13 x°G.+Иj.
00538: D6 00 00 00 41 68 3F 75 Ц...Ah?u	00538: 6A 00 68 E2 83 40 00 FF j.hвф@.я
00540: 40 00 FF 15 F8 60 40 00 @.я.ш`@.	00540: 15 6C B0 47 00 B9 4D 00 .l°G.MM.
00548: 33 C5 47 8D 05 62 86 40 3EGК.b+@	00548: 00 00 4B 6A 00 FF 15 1C ...Kj.я..

Variant 1

Variant 2

Figure 2: Polymorphic Malware Comparing “Backdoor.Win32.Shiz.pe” Shiz PE files:

2.6.5 Metamorphic Malware

Malware that adopts metamorphism alters itself as it propagates thus making the specimen hard to distinguish as variants are slightly different from the first, enabling the specimen to spread to further systems. Metamorphic malware relies greatly on the mutation algorithm employed to create mutations of that specimen and if this algorithm is not implemented correctly enumeration maybe used to identify possible irritations of the metamorphic engine. The inventive ways that malware mutates on machines means overall detection and removal of an infection on systems are difficult. Metamorphic code generators apply code obfuscation

techniques to produce structurally different versions of the same specimen of malware. **Figure 3** uses garbage code insertion resulting in a unique metamorphic variant of the same specimen.

ngvck98.asm	ngvck141.asm
<pre> : Win32.NGVCK by SnakeByte : : This Virus is created with : the Next Generation VCK by SnakeByte : to get a copy of this Kit : check www.kryptocrew.de/snakebyte/ .586p .model flat jumps .radix 16 extrn ExitProcess:PROC .data VirusSize equ (offset EndVirus - offset Virus) NumberOfApis equ 10d .code start: VirusCode: Virus: jmp labelblock1 add bx, 0 shl cx, 0 labelblock22: add eax, -1 inc eax jz UnMapFile mov dword ptr [ebp+MapAddress], eax clc ret UnMapFile: ; Unmap the file and store it to disk Call UnMapFile2 CloseFile: ; Close the file push dword ptr [ebp+FileHandle] jmp labelblock23 shr bx, 0 test cx, 0 labelblock56: Notagoodfile: </pre>	<pre> : Win32.NGVCK by SnakeByte : : This Virus is created with : the Next Generation VCK by SnakeByte : to get a copy of this Kit : check www.kryptocrew.de/snakebyte/ .586p .model flat jumps .radix 16 extrn ExitProcess:PROC .data VirusSize equ (offset EndVirus - offset Virus) NumberOfApis equ 10d .code start: VirusCode: Virus: jmp labelblock1 xor bx, 0 shr cx, 0 labelblock50: push dword ptr [ebp+MapAddress] pop ebx mov ebx, [ebx+3Ch] add ebx, dword ptr [ebp+MapAddress] mov edx, dword ptr [ebp+CheckSum] mov dword ptr [ebx+58h], edx NoCheckSum: mov ecx, dword ptr [ebp+InfCounter] add ecx, -1 jmp labelblock51 or cx, 0 add cx, 0 labelblock58: </pre>

Figure 3: Metamorphic Trojan w32 NGVCK (Next Generation Virus Creation Kit)

VX Heavens website lists various malware generators including NGVCK as used in Figure 1 above some others include:

- I. PS-MPC (Phalcon/Skism Mass-Produced Code generator)
- II. VCL32 (Virus Creation Lab for Win32)
- III. G2 (Second Generation virus generator)
- IV. MPCGEN (Mass Code Generator)

2.6.6 Garbage Code Insertion

Garbage code insertion doesn't change the behaviour of a program this technique adds benign program instructions to change the appearance of the programmes binary sequence (Xufang, et al., 2011; Konstantinou & Wolthusen, 2008). Various types of garbage codes come in the form of operand register (NOP) values as illustrated in Figure 3, code reordering through jumps and

equivalent instruction substitution however, and signature based malware detection can overcome this technique by deleting benign instructions before analysis.


<pre> 00401005 8BF0 MOV ESI,EAX 00401007 3E:8A00 MOV AL,BYTE PTR DS:[EAX] 0040100A 84C0 TEST AL,AL 0040100C 74 46 JE SHORT Test.00401054 0040100E 53 PUSH EBX 0040100F 3E:8F05 74F940 POP DWORD PTR DS:[40F974] 00401016 D3DB RCR EBX,CL 00401018 0FCB BSWAP EBX 0040101A 68 56104000 PUSH Test.00401056 0040101F 5B POP EBX 00401020 3E:8903 MOV DWORD PTR DS:[EBX],EAX 00401023 43 INC EBX 00401024 0FBDC2 BSR EAX,EDX 00401027 A9 46A978DC TEST EAX,DC78A946 0040102C 8BC2 MOV EAX,EDX 0040102E 52 PUSH EDX 0040102F B6 86 MOV DH,86 00401031 B3 27 MOV BL,27 00401033 B8 7CFAA17F MOV EAX,7FA1FA7C 00401038 EB 01 JMP SHORT Test.0040103B 0040103A 90 NOP 0040103B 0FBCC2 BSF EAX,EDX 0040103E 3E:C705 FC8841 MOV DWORD PTR DS:[4188FC],0 00401049 2D 210DE8B9 SUB EAX,B9E80D21 0040104E 69DA E577D49D IMUL EBX,EDX,9DD477E5 </pre>		<pre> 00401005 8BF0 MOV ESI,EAX 00401007 3E:8A00 MOV AL,BYTE PTR DS:[EAX] 0040100A 84C0 TEST AL,AL 0040100C 74 49 JE SHORT Test.00401057 0040100E 53 PUSH EBX 0040100F 3E:8F05 74F940 POP DWORD PTR DS:[40F974] 00401016 90 NOP 00401017 D3DB RCR EBX,CL 00401019 0FCB BSWAP EBX 0040101B 68 59104000 PUSH Test.00401059 00401020 5B POP EBX 00401021 3E:8903 MOV DWORD PTR DS:[EBX],EAX 00401024 90 NOP 00401025 43 INC EBX 00401026 0FBDC2 BSR EAX,EDX 00401029 A9 46A978DC TEST EAX,DC78A946 0040102E 8BC2 MOV EAX,EDX 00401030 52 PUSH EDX 00401031 90 NOP 00401032 B6 86 MOV DH,86 00401034 B3 27 MOV BL,27 00401036 B8 7CFAA17F MOV EAX,7FA1FA7C 0040103B EB 01 JMP SHORT Test.0040103E 0040103D 90 NOP 0040103E 0FBCC2 BSF EAX,EDX 00401041 3E:C705 FC8841 MOV DWORD PTR DS:[4188FC],0 00401044 2D 210DE8B9 SUB EAX,B9E80D21 00401049 69DA E577D49D IMUL EBX,EDX,9DD477E5 </pre>
--	---	---

Figure 4: Garbage Code Insertion: operand register (NOP) values insertion

2.6.7 Register Assignment Substitution

Mutation engines are used to exchange registers or memory variables in different instances of a virus. The technique attempts to overcome string signature detection, by converting the identical bytes in different generations. This technique does not change the behaviour of the code, but modifies the binary sequence of the code as illustrated in Figure 4 by reassigning EAX, EBX and EDX to EBX, EDX and EAX in that order.


<pre> 00401005 8BF0 MOV ESI,EAX 00401007 3E:8A00 MOV AL,BYTE PTR DS:[EAX] 0040100A 84C0 TEST AL,AL 0040100C 74 46 JE SHORT Test.00401054 0040100E 53 PUSH EBX 0040100F 3E:8F05 74F940 POP DWORD PTR DS:[40F974] 00401016 D3DB RCR EBX,CL 00401018 0FCB BSWAP EBX 0040101A 68 56104000 PUSH Test.00401056 0040101F 5B POP EBX 00401020 3E:8903 MOV DWORD PTR DS:[EBX],EAX 00401023 43 INC EBX 00401024 0FBDC2 BSR EAX,EDX 00401027 A9 46A978DC TEST EAX,DC78A946 0040102C 8BC2 MOV EAX,EDX 0040102E 52 PUSH EDX 0040102F B6 86 MOV DH,86 00401031 B3 27 MOV BL,27 00401033 B8 7CFAA17F MOV EAX,7FA1FA7C 00401038 EB 01 JMP SHORT Test.0040103B 0040103A 90 NOP 0040103B 0FBCC2 BSF EAX,EDX 0040103E 3E:C705 FC8841 MOV DWORD PTR DS:[4188FC],0 00401049 2D 210DE8B9 SUB EAX,B9E80D21 0040104E 69DA E577D49D IMUL EBX,EDX,9DD477E5 </pre>		<pre> 00401005 8BF3 MOV ESI,EBX 00401007 3E:8A1B MOV BL,BYTE PTR DS:[EBX] 0040100A 84DB TEST BL,BL 0040100C 74 48 JE SHORT Test.00401056 0040100E 52 PUSH EDX 0040100F 3E:8F05 74F940 POP DWORD PTR DS:[40F974] 00401016 D3DA RCR EDX,CL 00401018 0FCA BSWAP EDX 0040101A 68 58104000 PUSH Test.00401058 0040101F 5A POP EDX 00401020 3E:891A MOV DWORD PTR DS:[EDX],EBX 00401023 42 INC EDX 00401024 0FBDD8 BSR EBX,EAX 00401027 F7C3 46A978DC TEST EBX,DC78A946 0040102C 8BD8 MOV EBX,EAX 0040102E 50 PUSH EAX 0040102F B4 86 MOV AH,86 00401031 B2 27 MOV DL,27 00401033 BB 7CFAA17F MOV EBX,7FA1FA7C 00401038 EB 01 JMP SHORT Test.0040103C 0040103A 90 NOP 0040103B 0FBCC8 BSF EBX,EAX 0040103E 3E:C705 FC8841 MOV DWORD PTR DS:[4188FC],0 00401049 81EB 210DE8B9 SUB EBX,B9E80D21 0040104E 69D0 E577D49D IMUL EDX,EAX,9DD477E5 </pre>
--	---	---

Figure 5: Register Assignment Substitution

2.6.8 Code Transposition

Code transposition or code permutation modifies the program structure by reordering program instruction or code flow while retaining the execution flow through conditional or unconditional branches and can be applied to a single instruction level or an entire code block. Virtualization

platforms have been affected by this obfuscation method; malware authors utilize code transposition to hinder the reverse engineering analysis process (Coogan, et al., 2011). An obfuscator is comprised of a virtual machine that interprets the logic of that program (Rolles, 2009). Figure 6, illustrate code transposition by randomly shuffling instructions and then recovers the original execution order by inserting the unconditional branches or jumps.



Figure 6: Code Transposition: transposition based on Unconditional Branches

2.7 Approaches to Malware Analysis

Two approaches to malware analysis exist dynamic and static. The latter, static analysis is a popular method for identifying if a file is malicious or benign. Static analysis information about data flow and other statistical features are extracted without running the program. Reverse engineering procedures such as disassembling and decompiling are used to create a transitional representation of the binary code without code execution. Dynamic analysis, on the other hand requires that the malware binary is executed within a virtual machine environment. The runtime behaviour of that binary is monitored based on system call traces or API calls that the binary invoke (Rabek, et al., 2003).

2.7.1 Static & Dynamic analysis advantages & disadvantages

Static analysis has the potential to examine the execution structure of the binary code and observe pieces of code that as a rule do not execute (Gomes, et al., 2010). There are severe limitations with current static analysis approaches particularly when obfuscation techniques such as runtime packing/compression, garbage code insertion, code transposition/permutation and encryption are in place (Yason, 2007).

Classification techniques have been adopted by researchers to assign unknown malware to formerly identified malware classes. In some work, APIs and other dynamically extracted strings are used as features to detect malware (Ahmed, et al., 2009). Combination of both the spatial and the temporal features in runtime API calls are used to build a model and distinguish malware from benign files. Dependency graphs can be used to model the runtime behaviours (system calls) of PEs samples (Karbalaie, et al., 2012). Ghiasi et al (2012) used values of register contents in the runtime to represent the behaviour of each binary (Ghiasi, et al., 2012).

Dynamic approaches suffer from limitations, to begin with dynamic analysis may fail to observe the entire capabilities of a given malware sample because they may not cover its whole behaviour during that particular execution; dynamic analysis is also resource intensive. Time constraints, due to the analysis being resource intensive and volume of malware samples that are collected daily may result in each sample being executed for a short period of time which is not usually sufficient to observe the malware's behaviour. The main advantage of the dynamic approach is that behavioural features are insensitive to low-level mutation techniques, such as runtime packing or obfuscation given real information about the control and data flow.

On the basis of this literature review the research question for this work is both timely and relevant. Conducting, evaluating and analysing a malicious Windows PE file in a controlled virtual environment will determine the purpose and function of the specimen/artefact, its potential technical impact and the contribution to security education.

3 DESIGN

3.1 Introduction

The following framework includes a set of features which progressively investigate if a PE file is malicious. As research would indicate that there is no formal methodology to establish if a PE file is malicious. An analyst may use this framework as a foundation in the initial stages of malware detection and removal programs, but must adapt and augment this as emerging threats develop. Figure 7 below represents the general structure of such a framework.



Figure 7: Malware Dissection Analysis Framework

3.2 Fully-Automated Analysis

Fully-automated analysis is the simplest technique to evaluate a malicious file using fully-automated tools and online services such as FireEye developed for programme executables (PE) file analysis, the previously mentioned are intended to evaluate what a specimen does after execution on a system. Automated analysis tools produce reports detailing information such as registry keys used by the malicious program, file activity, and network traffic and mutex values. Automated analysis aids incident response through saving analysts time by analysing a specimen's capabilities, therefore the analysts can focus on specimens that require manual analysis.

3.3 Static Analysis

An analyst's investigation of a suspicious file will continue by examining the static properties of that specimen, this is a relatively swift process as the potentially malicious file is not executed. Static properties include hashes, packer signatures, header details, embedded resources and meta data. Analysing static properties can occasionally be adequate verification indicating that

a compromise has transpired. Additionally, this process determines whether an analyst should take a closer look at the specimen using thorough techniques conducive to dynamic analysis and where to focus the subsequent steps of the framework.

3.4 Dynamic Analysis

Reverse engineering code of a specimen adds valuable insight to the findings available after implementation of interactive behaviour analysis. Some characteristics of a specimen are simply impractical to examine without examining the code. Manual code reversing provides insights that include:

- Encrypted data decoding that's transferred or stored by the specimen under analysis.
- Determine what the malicious program's generation algorithm logic functions.

Debuggers, decompilers and disassemblers are utilised to aid the code reversing process, along with memory forensics. Reversing code is time consuming require patients and determination to achieve the end goal which is establish the specimens end agenda, therefore a controlled environment is used.

3.5 Interactive Behaviour Analysis

The interactive behaviour analysis of the framework requires for an isolated operating system being infected with the malicious specimen to monitor its behaviour during execution, gaining understanding of file structure, process, registry values and network activities. During this phase of analysis the analyst interacts with the malicious program, thus increasing coverage of the specimen interaction with the operating system instead of passively examining the specimen's interaction. An analyst may witness the specimen's endeavours as it attempts to contact a specific C2 server, that isn't reachable within the virtual lab. An analyst may mimic the system interactions in the virtual lab by replicating the experiment by altering Iptables, creating a fake DNS and creating a HTTP daemon which will respond to HTTP requests or even simulating the activities attacker's C2 to establish what the malicious program would do after it is able to connect to a C2 server.

3.6 Combining Analysis Stages

Regardless of platform, detection models based on a representative set of malicious and benign files using static analysis (Nissim, Moskovitch, Rokach and Elovici, 2014) have shown some advantages in their ability to generalize about the capabilities of malware. Static analysis improves the detection of malware with a reasonably high degree of probability however is it time consuming. Given the scale of data supplied when analysing malware, similarity analysis techniques from Jang et al. (2011) have shown some promise also when used for sorting and clustering data on a large-scale. While many methodologies may be employed in the discovery and remediation of malware the perfect digital crime scene rarely exists since logs are deleted,

files overwritten and hardware damaged (Malin, et al., 2012). Current techniques rely upon signature-based detection of malicious code that have already been observed. Although sets of heuristics and rules may be adapted to search for the operational characteristics of malwares in unknown files, this technique is also time-consuming and in most cases, costly. Additionally, and perhaps more obviously, any unknown malware will have unique characteristics or combination thereof and will not be detected (Nissim, Moskovitch, Rokach and Elovici, 2014). Overcoming the limitations in these techniques some research in the area of 'non-signature based' dynamic analysis has shown high detection and low false positives (Shahzad, Shazad and Farooq, 2013). Although not widely used in the industrial case, this technique uses genetic foot printing which compares benign processes to malicious ones maintained within the kernel of an operating system. As important as detection, analysis and remediation of malware are, the creation of a suitable signature is also important since this must be fed back into the antivirus system as a defence.

3.7 Signature Creation

It is recommended that malware signature authors should avoid targeting cryptography or packer library code since this is often reused and has been shown to have a high false positive rate (Schiffman, 2010; Beaucamps, 2007). Code that is packed, repacked or compressed is problematic for signature detection, such files require decompression or dumping prior to scanning (Taha, 2007; Szor, 2005). Typically, unpackers and emulators are used by anti-virus engines to get the files to a dumped or decompressed condition prior to file scanning. However, a unique packer can identify a specific malware family that might be under analysis. Any change to the specimen will result in a unique variant within that malware family but the specimens real semantics are unchanged the intent still exists (Christodorescu & Jha, 2004). If data has been obfuscated or compressed, this is also not a viable candidate for signature generation. For example, in file hashes using MD5, a one byte change in the data may alter the compressed or obfuscated codes hash value. Since bytes of code can be changed simply by different data or keys, the possibility that the original identification data will not be present in other variants of that malicious specimen increases (Khazan et al., 2003). As regards current techniques used in signature generation, several of the following approaches suggest themselves as valid detection methods. These include hash signature, string signature and file diffing.

3.7.1 Hash Signatures

A hash signature is obtained through a hash function that is a mathematical function or procedure that converts data into a single value; MD5 and SHA-1 are the most common hash functions. Hash functions are tamper proof, as illustrated in Figure 6: a data block is run through a hash function and a byte changed in that same block of data, then the rehashed hash value will be entirely different. (Kornblum, 2006).

Your Hash: **d7ad1763be326a1d4183457307802911** Your Hash: **8f5f74a1354835d2024248cace023fc0**
Your String: MaliciousSpecimen00 Your String: MaliciousSpecimen01

Figure 8: A Single Byte Change from "00" To "01" Results In A Unique Hash Signature

It is possible to use ClamWin AV to create MD5 based signatures using two characteristic attributes of a malware specimen. The first is the file size in bytes and the second is the MD5 hash. The sigtool located in the "bin" directory of the installation folder is a feature of ClamWin AV which may be adopted to produce a MD5 signature.

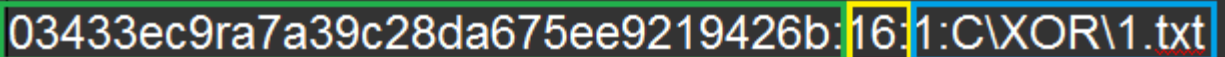
A screenshot of a text string representing an MD5 hashed signature. The string is "03433ec9ra7a39c28da675ee9219426b:16:1:C\XOR\1.txt". The first part, "03433ec9ra7a39c28da675ee9219426b", is highlighted in green. The second part, "16", is highlighted in yellow. The third part, "1:C\XOR\1.txt", is highlighted in blue.

Figure 9: MD 5 Hashed Signature Portions

A colon ":" separates signature features within ClamWin AV. In Figure 9 above the portions of a MD5 hashed signature are illustrated, firstly the green portion represents the MD5 hash value of the signature while the yellow portion is the files size and the blue portion represents the output or file location. The output or file location should uniquely identify that malware specimen, colons ":" are special characters that aren't permitted in the signatures database of ClamWin AV but are necessary for signature creation. MD5 hashed signatures are saved with an .hdb extension

A hash values function is undermined if the malware specimen is not static, if a single byte is augmented in the executable that hash signature becomes obsolete as illustrate in Figure 8 above. A Remote Access Trojan (RAT) also referred to as creep-ware is an example of an executable that never changes it's code base or block however, the executables data does change as it incorporate the end user's IP address and therefore the data sections would differ. Figure 10 illustrates sectional hashing of the Pandora RAT, a sectional hashing may be utilised on that executables code base creating a hash signature for Pandora RAT. In Figure 10 the first portion in green marks the PE section, the yellow portion represents the MD5 hash and the blue section represents the malware name in this instance Pandora.

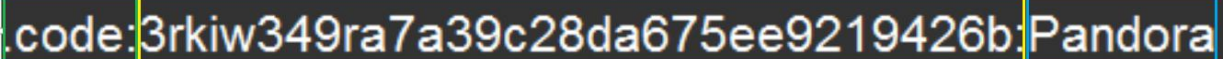
A screenshot of a text string representing a sectional hash signature for Pandora RAT. The string is ".code:3rkiw349ra7a39c28da675ee9219426b: Pandora". The first part, ".code:", is highlighted in green. The second part, "3rkiw349ra7a39c28da675ee9219426b", is highlighted in yellow. The third part, "Pandora", is highlighted in blue.

Figure 10: Sectional Hashing of Pandora RAT and Hash Base Signature Creation

ClamWin AV requires that the signature be saved as an .mdb file to be the accepted format. Once again colons ":" are reserved characters within the ClamWin AV signature database.

Hashing can be utilise for identifying files that were modified through fuzzy hashing. A Fuzzy hash is an amalgamation of context piecewise triggered hashing and recursive computing. This

method is utilised for identifying files that have been modified or that have new data inserted or data deleted in an attempt to avoid detection.

String signature are based on byte sequences that are present in a data stream or in a file. String signature efficacy is derived from the accuracy provided from detecting a sequence of bytes forming distinctive pattern or common characteristic which exists in manifold variants of malware from the one malware family. String signatures could be formed from any data type of data such as data enclosed in a data stream of an executable i.e. XOR cipher (Griffin, et al., 2009).

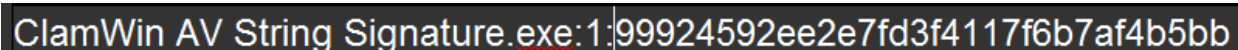
A screenshot of a ClamWin AV scanner output showing a string signature for a file named "ClamWin AV String Signature.exe". The signature is displayed as "1:99924592ee2e7fd3f4117f6b7af4b5bb". The "1" indicates the file type (PE executable) and the hexadecimal string "99924592ee2e7fd3f4117f6b7af4b5bb" represents the opcode sequence.

Figure 11: ClamWin AV String Signature

Figure 11 represent output shown by ClamWin AV scanner starting with the malware name “ClamWin AV String Signature”, the second portion “1” identifies the file type in this instance a PE executable is for the ClamAV engine to know the file type of the scanned file. The value '1' is for the engine to scan portable executable files a “0” value would indicate any file type. The final portion of the string signature is hexadecimal version "99924592ee2e7fd3f4117f6b7af4b5bb" is of the opcodes.

3.7.2 File Diffing

Semi -automated and manual analysis methods maybe utilised for detecting blocks of code that are exist in multiple variants of a malicious specimen. Binary diffing is method adopted to compare multiple executables for similarities. Binary diffing manually requires reviewing of disassembly output for multiple files and then recording sections of code that exist in multiple files, on identification of code blocks a side-by-side evaluation is undertaken checking similarities. If code blocks have numerous similarities then it is possibility a candidate for string signature creation. Binary diffing using the semi-automated method involves grouping files into sets that similarities exit in, this is achieved by using a disassembler to extract assembly code into a text output, this output is then diffed, creating specific sets of diffs that match at this point the code sections are manually analyse to detect matching sequences (Malin, et al., 2012)

Diffing can be utilised to identify code sections that use wildcards permitting ClamWin AV scanner to disregard code section by skipping bytes or byte ranges in code that aren't static in multiple variants. The addition or removal of code or data or the insertion of garbage or junk code may result in a non-static byte or bytes within code. Manipulating file section alignment of a PE file format through the use of offsets including jmps, call sub-routine or call an api or others that cmp, mov, sub and add instruction referencing offsets, may circumvent the string signatures that focuses on specific file section instructions (Malin, et al., 2012). Best practice is

to include wildcards for function offsets when developing signatures even if static in different variants of the same malware family.

In Figure 12 “DiffingWildcardCharacters_01” and “DiffingWildcardCharacters_02” executable are used to illustrate how string signatures can be broken using file section offsets, the only disparity, between the code blocks is the offset address of “dword” (DiffingWildcardCharacters_01 highlight in the blue and DiffingWildcardCharacters_02 in yellow boxes) which is compared against ebx as the output string is a different length there is a difference in file alignment.

```
DiffingWildcardCharacters_01.exe
start: 0x401010  lenght: 0x14
2D 1C 50 55 31 60 cmp ebx, of-set dword_40318C
2D A6 4D CA 47 44 jnd loc_401848
81 10 9F C8 AE mov esi, 400000h
4C 13 64 lea edi [ebp-var_20]

2D 1C 50 55 31 60 1D A6 4D CA 47 44 81 10 9F C8 AE 4C 13 64

DiffingWildcardCharacters_02.exe
start: 0x401010  lenght: 0x14
2D 1C C6 55 31 60 cmp ebx, of-set dword_409140
2D A6 4D CA 47 44 jnd loc_401848
81 10 9F C8 AE mov esi, 400000h
4C 13 64 lea edi [ebp-var_20]

2D 1C C6 55 31 60 1D A6 4D CA 47 44 81 10 9F C8 AE 4C 13 64

ClamAV Signature format:
Both DiffingWildcardCharacters dot exe:1:2DC1*****1DA64DCA474481109FC8AE4C1364

Yara Format:
Rule Both_DiffingWildcardCharacters_dot_exe
{
strings:
signature - {2D C1 ?? ?? ?? ?? ?? 1D A6 4D CA 47 44 81 10 9F C8 AE 4C 13 64}
condition:
signature
}
```

Figure 12: Diffing with Wildcard Characters When Section Offsets Are Present

4 IMPLEMENTATION

4.1 Lab design, environment configuration and analysis report structure.

The primary objective of malware analysis is gain insight into what a malware specimen's intent is and gain understanding of behaviour and internal configuration through analysis of the specimen's code. Hence establishing:

- Characteristic and capabilities of the malicious specimen under analysis
- The damage potential.
- Indicators of compromise (IOC) which may be used to identify future compromises or activity used within memory in the network or at rest in file system. Observed IOCs can be incorporated into defence systems and incident response process.

The course of action taken during malware analysis requires executing the specimen executable under investigation in a controlled isolated secure environment. The dynamic analysis methodology allows you to determine the malware behaviour and how it interacts with the network, file system, registry and others.

VMware Workstation will host two virtual machines, one machine running REMnux v5. The second machine will be running Windows 7 32 Bits. As lustrated in Figure 13 below, the REMnux virtual machine will be used as a communication gateway, proxy and DNS server. Thus allowing for interception of network communications initiated from the infected windows machine.

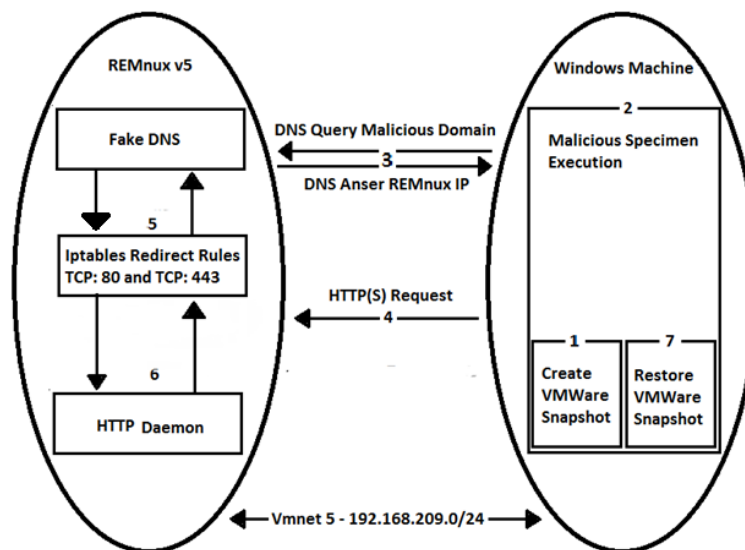


Figure 13: Lab Design and Connectivity

Figure 13 illustrates how the virtual REMnux machine and the Windows 7 machine communicate as DNS requests are redirected to the REMnux virtual machine. The virtual REMnux machine will intercept requests even if the malicious specimen uses a hardcoded IP address instead of DNS. The REMnux machine's iptables will be configured to intercept traffic on TCP port 8080 and port TCP 80 or 443.

4.1.1 REMnux Machine Configuration

Configuration REMnux as follows: (Step by step illustration contained in the appendices)

1. Network adapter settings defined on VMware Workstation in a virtual custom network, VMnet5.
2. Static IP address defined.
3. DNS requests to be answered by fake DNS.
4. HTTP daemon will reply to HTTP requests.
5. Iptables are configured to redirect HTTP and HTTPS traffic to port TCP.
6. Run Wireshark to capture all the networking traffic, allowing for the creation of malware signatures.

The commands to perform steps 3,4,5 and 6 are located in the appendix:

4.1.2 Windows 7 Machine Configuration

On the Windows 7 machine the set up steps are as follows: (Step by step illustration contained in the Appendices - REMnux Machine Configuration)

1. Identify network adapter settings in VMware to be in REMnux's virtual network.
2. Configure IP address in the same range as the REMnux
3. Configure the DNS server to point to the REMnux
4. Define the default gateway as being the REMnux
5. Test the network settings
6. Create a VMware snapshot
7. Move the malware sample to the machine
8. Start necessary tools (if needed)
9. Execution of the malicious specimen – MaliciousSpecimen01.exe

4.2 Malware Analysis Report Structure

The intended malware analysis report structure will detail investigation components of the aforementioned analysis and will refer to specific tools that can be helpful in correlating certain data such as functions that create a service on execution of the specimen.

4.3 Malicious Specimen Selection

The malicious specimen selected for this thesis is a Programme Executable (PE) file which refers to executable image and object files in the Windows operating systems family. In this instance the PE file is used as the attack vector by means of process injection. Process injection is technique used by PE malware for a number of reasons that include running without a process, positioning user mode hooks for a form grabber or rootkit and to bypassing antivirus via injecting white listed processes. The most frequent used technique of process injection is DLL Injection, which is popular due to the unproblematic nature of dropping a file to disk. A program can simply drop a DLL to the disk and subsequently use "CreateRemoteThread" to call "LoadLibrary" in the target process, the loader takes care of the remaining process. PE Injection is generally favoured over DLL Injection by malware, because it does not require dropping any files to the disk. PE's run in memory and make use of two structures Import Address Table (IAT) and Base Relocation Table Reloc (Reloc).

Within IAT on loading a DLL into memory that DLL isn't guaranteed to be loaded to the same address each time, application makes use of an IAT to deal with this. The IAT permits for addresses of DLL functions to get set by the PE loader, without modifying the code of the application. Doing so by arranging all calls to DLL functions point to a jump in the processes' own jump table, the IAT then allows the address the jumps targets to be found and altered by the PE loader. The "Reloc" table also make it possible for an application not to load at the same address each time this, isn't an issue as the application uses relative addressing. However, as absolute addresses will require change if the process base address changes, whenever an absolute address is used, it must be easily located. The "Reloc" is a table of pointers to every absolute address applied in the code. During process initialization, if the process is not being loaded at its base address, the PE loader will modify all the absolute addresses to work with the new base address.

The "IAT" and "Reloc" table stay in memory as soon as the process initialization has finished, making for a suitable technique to injecting a process. With the capability to load at any base address and use DLL's at any address, the process is capable of getting its current base address and image size from the PE header and then copy itself to any region of memory in almost any process.

The entire procedure can be broken into stages as follows.

1. Get the current images base address and size (usually from the PE header).
2. Allocate enough memory for the image inside the processes own address space (VirtualAlloc).
3. Have the process copy its own image into the locally allocated memory (memcpy).
4. Allocate memory large enough to fit the image in the target process (VirtualAllocEx).

5. Calculate the offset of the reloc table for the image that was copied into the local memory.
6. Iterate the reloc table of the local image and modify all absolute addresses to work at the address returned by `VirtualAllocEx`.
7. Copy the local image into the memory region allocated in the target process (`WriteProcessMemory`).
8. Calculate the remote address of the function to be executed in the remote process by subtracting the address of the function in the current process by the base address of the current process, then adding it to the address of the allocated memory in the target process.
9. Create a new thread with the start address set to the remote address of the function (`CreateRemoteThread`).
10. In some cases once the image is executed in the remote process, it may have to fix its own IAT so that it can call functions imported from DLLs, however; DLLs are usually at the same address in all processes, so this wouldn't be necessary.

Revealing features to identify malware injection include, processes allocating memory inside another process such as `VirtualAllocEx`, `NtMapViewOfSection` and `NtAllocateVirtualMemory`, particularly if the memory is allocated with the `PAGE_EXECUTE` flag. Other identification characteristics might include a process that sets the `PAGE_EXECUTE` flag of a memory region in another process such as `NtProtectVirtualMemory` or `VirtualProtectEx`. A process that creates a thread in another process `NtCreateThread`, `RtlCreateUserThread`, `CreateRemoteThread`, if that thread points to code within a memory region, which was also allocated by the same process or if a process appends code to shared sections.

5 EVALUATION & TESTING

5.1 Introduction

In chapter 3 an analysis framework was proposed to investigate behaviour and inner workings of a PE file, chapter 4 establishes the bases for creating a lab specifically designed for evaluation and testing of PE files.

5.2 Malware Analysis Report

Defending an increasingly vulnerable perimeter with modern tools has become more difficult. In conjunction with this, cyber-crime is a lucrative business and therefore not surprising the level of cybercrime devoted to the Internet has grown significantly (Kaspersky, 2013; Trend, 2014; Verizon, 2014). For example, in just a few years, Distributed Denial-of-Service (DDoS) attacks have escalated from dozens to hundreds of gigabits per second as a result of increasingly sophisticated malware (Akamai, 2012). Threats to network and information security have existed for quite some time and given the reported increases in scale and complexity of attacks in recent years, some care must be taken when presenting enterprises with the considerable challenges posed by security reports. The significance of this report is recognized by decision makers engaged in the planning and purchasing of products and services aimed low cost solutions and time savings. As it stands, many enterprises struggle with the rapid pace of development in traditional perimeter defence solutions and the concordant increase in risks posed. Flexibility across a broad set of defence capabilities is required where timely, proactive defences adapt and evolve with ever changing risks and unknowable future threats rather than the expensive process of dealing with security events after the fact. Several elements to the malware analysis report include, but are not limited to, contextual information such as the date the file was discovered, a static analysis of the malware specimen and its characteristics along with the dynamic analysis of the malware's activity within the host system which may incorporate clues from static and behavioural analysis.

5.3 Specimen Selection & Testing

As a proof of concept the malicious specimen "MaliciousSpecimen01.exe" has been evolved to bypass anti-virus detection, such files get executed once the users runs the PE file and infects the system. A validation test is conducted as illustrated in Figure 14 and 15 to verify that no hash signature or Authenticode are associated with the specimen under analysis. Automated analysis within the Windows 7 machine and within the REMnux machine indicated that no malicious files were identified.

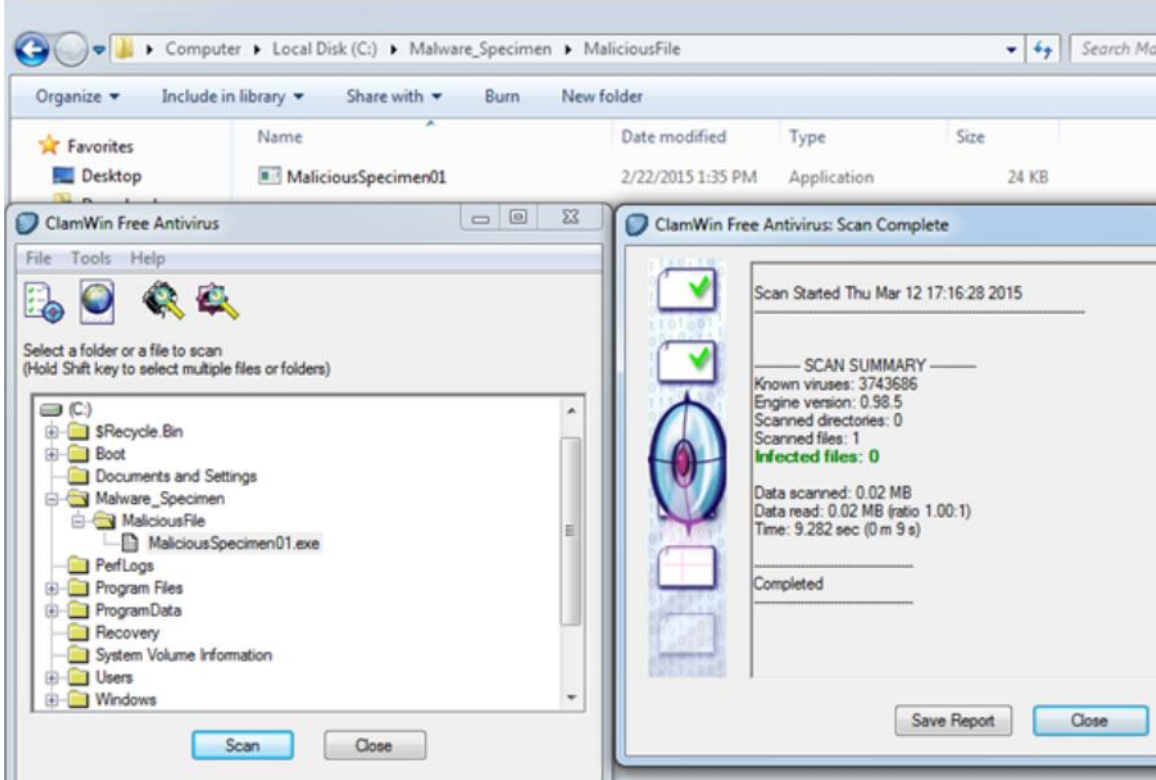


Figure 14: Windows 7 ClamWin MaliciousSpecimen01.exe scan results

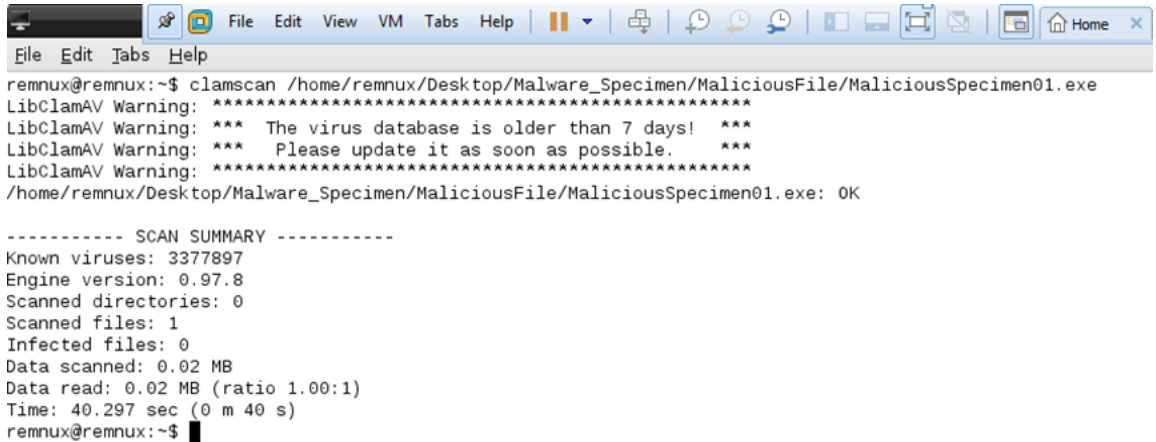


Figure 15: REMnux ClamAV MaliciousSpecimen01.exe scan results

A PE files integrity and origin should be authenticated via its Authenticode which is a digital signature applied to the file using the Public-Key Cryptography Standards (PKCS) #7 signed data and X.509 certificates to connect that PE file binary to a specific software publisher. Authors of malicious software may incorporate a digit signature the malware that in turn can be examined by analysts which gives understanding, insight can context to an incident. Analysts may use the digit signature in active defence systems as an indicator of compromise (IOC). Windows PE file

signatures are located in a specified certificate table entry in Optional Header Data Directories. The location of the signature is stored within the PE header's Optional Header structure's within Security field.

Pyew is used to establish if the PE file under analysis includes an embedded signature. On loading the specimen into Pyew, check the size of the "IMAGE_DIRECTORY_ENTRY_SECURITY" field. A greater than zero value indicates that the file includes an embedded signature. Loading the PE file into Pyew and entering the command as illustrated in Figure 16 below "pyew.pe.OPTIONAL_HEADER.DATA_DIRECTORY". As also illustrated in Figure 16 below the size of "IMAGE_DIRECTORY_ENTRY_SECURITY" is zero.

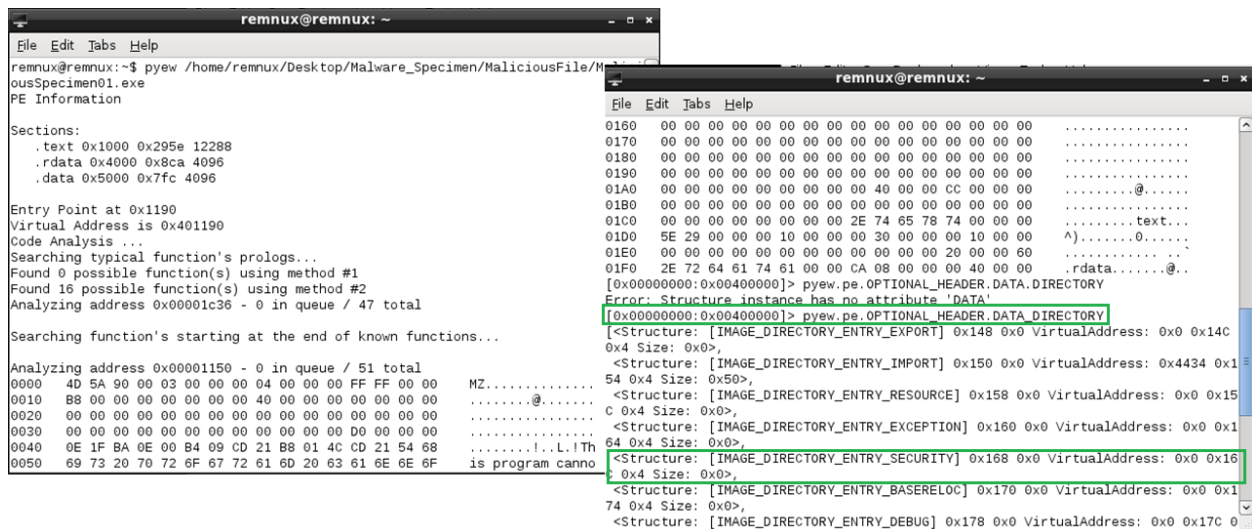


Figure 16: Pyew - PE file embedded signature test

5.4 Static Analysis

In static analysis a non-executing specimen of the malware is logged for observation. A cryptographic hash of the file using PHash will be the identification pattern that uniquely identifies the specimen. This is important since now this fingerprint can be drawn on as a means to check online services such as Virus Total to validate if malicious intent is present in a file. Figure 17 shows how PHash automatically generates a multiple file hash using different hashing algorithms.

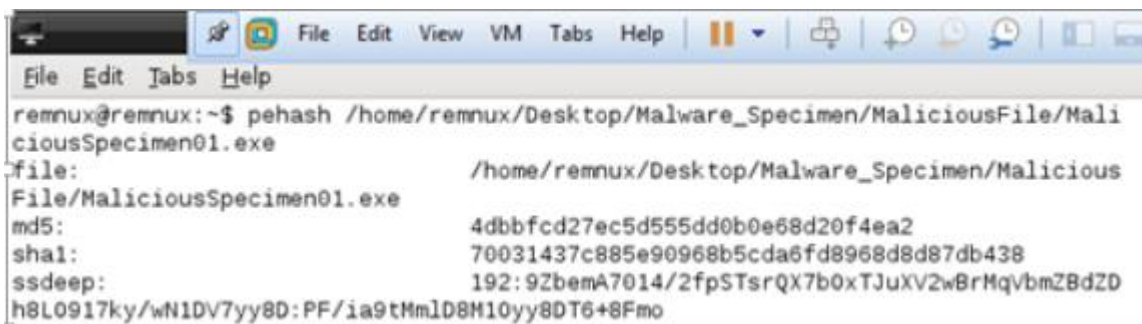


Figure 17: PEhash generating the MD5, SHA1 and SSDEEP hash of the malware specimen.

Online services such as VirusTotal and Comodo Valkyrie permit uploading of suspected malware files. This analysis is important since anti-virus programs block the installation of malware on the host machine. When a suspected malware file has been uploaded, it is shared with other service providers. If the malware is recognized anti-virus vendors will typically post analysis describing its characteristics, signatures and instructions for removal. Figure 18 shows the “No comments” tagged under Virustotal which indicates that is specimen has not been observed previously.

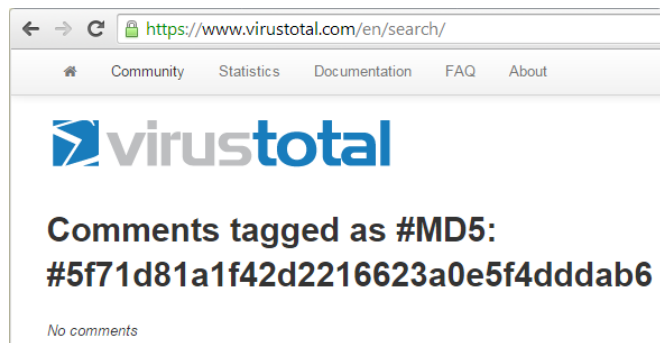


Figure 18: VirusTotal analysis Result

Further analysis illustrated in Figure 19 where FireEye a commercially available detection tool did not verify that the specimen exist however generated a “Malicious Alert” indicating that there was activity launching a new process which should warrant further analysis.

Process	Started	C:\Users\Administrator\AppData\Local\Temp\MaliciousSpecimen01.exe Parentname: C:\Windows\SysWOW64\cmd.exe Command Line: "C:\Users\ADMINI~1\AppData\Local\Temp\MaliciousSpecimen01.exe" MD5: 4dbbfcd27ec5d555dd0b0e68d20f4ea2 SHA1: 70031437c885e90968b5cda6fd8968d8d87db438
Malicious Alert	Generic Process Launch Activity	Message: Startup behavior anomalies observed Detail: A new process has been launched

Figure 19: FireEye Report

Pescan is used to understand the behaviour of an executable as shown in Figure 20. Given that the evidence here suggests that the file has high entropy, it is possible that the file maybe packed indicating that a further analysis is warranted. This requires the utilisation of an unpacker such as upx as illustrated in Figure 21 which when run identifies the packer used as upx 3.07. At this point the file is once again run through pescan as illustrated in Figure 22, which is important as modification from runtime compression are removed thereby facilitating a true picture of the intent of the PE file under investigation.

```

remnux@remnux:~$ pescan /home/remnux/Desktop/Malware_Specimen/MaliciousFile/MaliciousSpecimen01.exe
file entropy:          7.284349 (probably packed)
fpu anti-disassembly: no
imagebase:            normal
entrypoint:          fake
DOS stub:            normal
TLS directory:       not found
section count:       3
UPX0:                suspicious name, zero length, self-modifying
UPX1:                suspicious name, self-modifying
UPX2:                suspicious name, small length
timestamp:           normal

```

Figure 20: Pescan is used to understand the behaviour of an executable

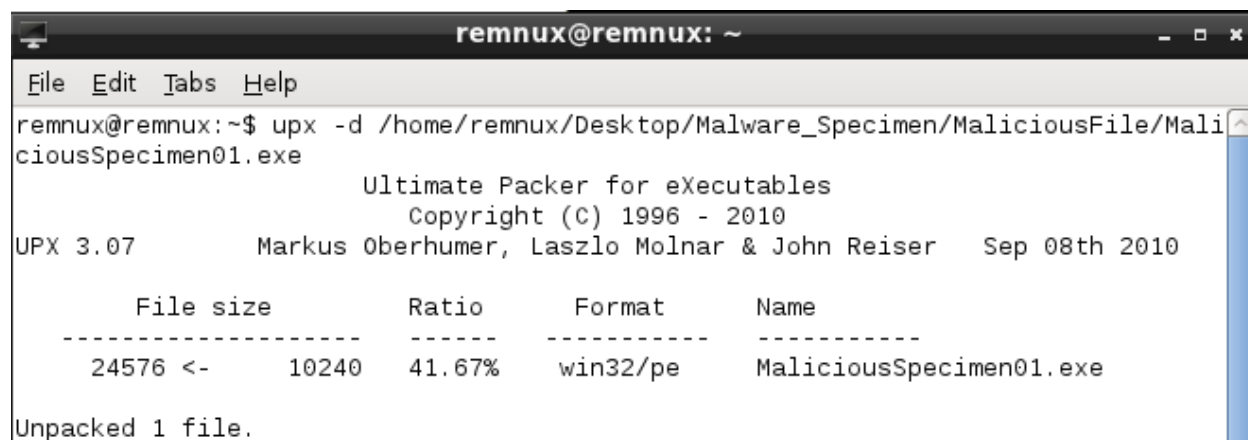


Figure 21: upx unpacker

```

remnux@remnux:~$ pescan /home/remnux/Desktop/Malware_Specimen/MaliciousFile/MaliciousSpecimen01.exe
file entropy:          4.234411 (normal)
fpu anti-disassembly: no
imagebase:            normal
entrypoint:          normal
DOS stub:            normal
TLS directory:       not found
section count:       3
.text:               normal
.rdata:              normal
.data:               normal
timestamp:           normal

```

Figure 22: pescan after unpacking

Next, the strings command is run against the malicious binary to display UNICODE and ASCII strings that may be rooted inside the file, disclosing information relating to the binary functionality. Such information might include functions names that interact with network, registry, etc as illustrated in Figure 23 where a network related URL string is disclosed.

```
remnux@remnux: ~  
File Edit Tabs Help  
remnux@remnux:~$ pestr --net /home/remnux/Desktop/Malware_Specimen/MaliciousFile  
/MaliciousSpecimen01.exe  
http://www.maliciousspecimen01.com
```

Figure 23: pestr output of a network related string

As illustrated in Figure 24, if it is the case that the strings output does not yield any important results since obfuscated techniques are adopted such as packing or encryption on creation of the malware variant then the unpacking process is required.

The final phase in the static analysis process requires using Readpe to analyse headers of the Win32 PE file. Obtaining evidence from the PE file header may possibly divulge information regarding API calls imported and exported by the malicious executable. Regular extensions for Windows PE files include .exe, .dll, .sys, .drv, cpl, .ocx. When binaries are compiled static linking or dynamic linking can be performed. The former static linking specifies helper functions required to execute the binary are inside the binary, dynamic linking runs and resolves pointers to the helper functions during run time. Binaries contain library dependencies which can be inspected, thus inferring what type of functionality present within the executable during static analysis. These dependencies are contained within the Import Address Table (IAT) portion within the PE structure so that the Windows loader (ntdll.dll) knows which functions and dll's are needed for the binary to operate correctly.

Identification of dll's and functions can be achieved through the use of Readpe. Ordinarily binaries contain large numbers of imported functions. As illustrated in Figure 24, the imported functions is undersized which suggests the binary is somehow packed, obfuscated or encrypted.

```
Imported functions  
  
KERNEL32.DLL  
LoadLibraryA  
GetProcAddress  
VirtualProtect  
VirtualAlloc  
VirtualFree  
ExitProcess  
  
ADVAPI32.dll  
CreateServiceA  
  
WININET.dll  
InternetOpenA
```

Figure 24: Readpe results before unpacking of MaliciousSpecimen01.exe

The reality is that the PE file MaliciousSpecimen01.exe contains an enormous number of imported functions some of which are listed in Figure 25. The malicious specimen calls upon "LoadLibraryA" and "GetProcAddress" which are Kernel32.dll windows functions which is significant as both functions are popular among packed malware authors as the aforementioned functions permit loading of the binary and obtaining access to other functions within Windows operating system.

	HeapReAlloc
	GetProcAddress
	LoadLibraryA
	MultiByteToWideChar
	GetStringTypeW
ADVAPI32.dll	
	CreateServiceA
	StartServiceCtrlDispatcherA
	OpenSCManagerA

Figure 25: Readpe results after unpacking MaliciousSpecimen01.exe

Static analysis enable the analyst focus on specific element within the malicious PE to examine in detail using dynamic analysis.

5.5 Dynamic Analysis

Dynamic analysis is the practice of characterizing constant change, activity, or progress within the host system. The initial phase to perform dynamic analysing of this specimen is to open it through Vivisect Disassembler to examine the imported functions list that where observed during static analysis as illustrated in Figure 26. PE files developed with malicious intent often utilize Windows service functions in the list and provide little information since multiple service functions are regularly imported by windows executable. However, "OpenSCManager" and "CreateService" stand out, as both functions reveal that this specimen creates a service to ensure that it will run the next time the Windows 7 machine is restarted. The "StartServiceCtrlDispatcherA" import suggests this file is really a service.



Address	Library	Function
0x00404000	advapi32	CreateServiceA
0x00404004	advapi32	StartServiceCtrlDispatcherA
0x00404008	advapi32	OpenSCManagerA

Figure 26: Vivisect Disassembler Imports

Further evaluation of this specimen as illustrated in Figure 27 indicates that calls to "InternetOpenA" and "InternetOpenUrlA" are occurring and possibly connecting to a URL containing malicious content which may result in a system compromise.

Imports		
Address	Library	Function
0x004040c0	wininet	InternetOpenUrlA
0x004040c4	wininet	InternetOpenA

Figure 27: Vivisect Disassembler Import

Using Vivisect the main functions location at "0x00401000" calls one other function as showing in Figure 28. The code initiates by calling "StartServiceCtrlDispatcherA" at "0x00401028". By referencing the MSDN documentation it is possible to establish that this function is used by a program to start a service instantaneously. The "StartServiceCtrlDispatcherA" function identifies the service control function called by the service control manager, in this instance "sub_00401040" which, is called after "StartServiceCtrlDispatcherA". The first portion of code is known as book-keeping code necessary for programme to run as a service.

```
.text:0x00401000 Segment: .text (10590 bytes) FIXME PERMS
.text:0x00401000
.text:0x00401000 FUNC: int cdecl sub_00401000( ) [2 XREFS]
.text:0x00401000
.text:0x00401000 Stack Variables:
.text:0x00401000         -4: int local14
.text:0x00401000         -8: int local18
.text:0x00401000        -12: int local12
.text:0x00401000        -16: int local16
.text:0x00401000
.text:0x00401000 83ec10         sub esp,16
.text:0x00401003 8d442400      lea eax,dword [esp]
.text:0x00401007 c744240030504000 mov dword [esp + local16],loc_00405030
.text:0x0040100f 50           push eax
.text:0x00401010 c7442400840104000 mov dword [esp + local12],sub_00401040
.text:0x00401018 c744240c000000000 mov dword [esp + local18],0
.text:0x00401020 c7442410000000000 mov dword [esp + local14],0
.text:0x00401028 ff1504404000 call dword [advap132.StartServiceCtrlDispatcherA_00404004] ;advap132.StartServiceCtrlDispatcherA(local16)
.text:0x0040102e 6a00         push 0
.text:0x00401030 6a00         push 0
.text:0x00401032 e8090000000 call sub_00401040 ;sub_00401040()
.text:0x00401037 83c418      add esp,24
.text:0x0040103a c3          ret
```

Figure 28: Vivisect Disassembler Function Dependencies

Examination of "sub_00401040" function reveals, that the first function call is "OpenMutexA" at "00x00401052" which is noteworthy as this call tries to attain a handle to mutex "1024" and if successful the programme exits as illustrated in Figure 29.

```
.text:0x00401040
.text:0x00401040 FUNC: int cdecl sub_00401040( ) [2 XREFS]
.text:0x00401040
.text:0x00401040 Stack Variables:
.text:0x00401040        -1012: int local1012
.text:0x00401040        -1016: int local1016
.text:0x00401040        -1020: int local1020
.text:0x00401040        -1024: int local1024
.text:0x00401040
.text:0x00401040 81ec00040000 sub esp,1024
.text:0x00401046 6848504000 push loc_00405048
.text:0x0040104b 6a00         push 0
.text:0x0040104d 6801001f00 push 0x001f0001
.text:0x00401052 ff1528404000 call dword [kernel132.OpenMutexA_00404028] ;kernel132.OpenMutexA(0x001f0001, 0, 0x00405048)
.text:0x00401058 85c0         test eax,eax
.text:0x0040105a 7408         jz loc_00401064
```

Figure 29: Vivisect Disassembler Function Dependencies

The code above generates a mutex "1014", which is designed to warrant that one copy of the malicious specimen is executed on the operating system at time. If a copy of the specimen is

running, then the original "OpenMutexA" call has been successful therefore, the programme would exit.

Subsequently after the "OpenMutexA" call as illustrated in Figure 30, "OpenSCManagerA", is called, opening the service control manager handler in order to modify or add services. The next the GetModuleFileNameA" function is called, returning a full pathname to a current running loaded DLL or executable. A new service is created by "CreateServiceA" if no service already exists

```

text:0x0040107a ff1508404000 call dword [advapi32.OpenSCManagerA_00404008] ;advapi32.OpenSCManagerA(0,0,3)
text:0x00401080 8bf0 mov esi,eax
text:0x00401082 ff1534404000 call dword [kernel32.GetCurrentProcess_00404034] ;kernel32.GetCurrentProcess()
text:0x00401088 8d44241c lea eax,dword [esp + 28]
text:0x0040108c 68e8030000 push 1000
text:0x00401091 50 push eax
text:0x00401092 6a00 push 0
text:0x00401094 ff1518404000 call dword [kernel32.GetModuleFileNameA_00404018] ;kernel32.GetModuleFileNameA(0,0xbfb00c18,1000)

```

Figure 30: Vivisect Disassembler Function Dependencies

Windows time structure manipulation is evident as per Figure 31 with "CreateWaitableTimerA" function call, the time structure has individual fields covering second, minute, hour, etc. used to stipulate a specific time. The executable calls "CreateWaitableTimerA" activating the timer and if the timer is active then "SetWaitableTimer" is called stopping the function. The "WaitForSingleObject" function checks the present status of the specific object. If the object state is "none signalled" no thread waiting on this event will be released, the calling thread enters dormant state until the object is signalled or the time-out interval elapses.

```

0x004010e5 ff1514404000 call dword [kernel32.SystemTimeToFileTime_00404014] ;kernel32.SystemTimeToFileTime(local1024,0xbfb00c18)
0x004010eb 6a00 push 0
0x004010ed 6a00 push 0
0x004010ef 6a00 push 0
0x004010f1 ff1510404000 call dword [kernel32.CreateWaitableTimerA_00404010] ;kernel32.CreateWaitableTimerA(0,0,0)
0x004010f7 6a00 push 0
0x004010f9 6a00 push 0
0x004010fb 6a00 push 0
0x004010fd 8d542420 lea edx,dword [esp + 32]
0x00401101 8bf0 mov esi,eax
0x00401103 6a00 push 0
0x00401105 52 push edx
0x00401106 56 push esi
0x00401107 ff151c404000 call dword [kernel32.SetWaitableTimer_0040401c] ;kernel32.SetWaitableTimer(kernel32.CreateWaitableTimerA(0,0,0),0xbfb00c18,0,0,0)
0x0040110d 6aff push 255
0x0040110f 56 push esi
0x00401110 ff152c404000 call dword [kernel32.WaitForSingleObject_0040402c] ;kernel32.WaitForSingleObject(kernel32.CreateWaitableTimerA(0,0,0),0xffffffff)
0x00401116 85c0 test eax,eax
0x00401118 7521 jnz loc_0040112b

```

Figure 31: Vivisect Disassembler Function Dependencies

The "InternetOpen" function initiates a connection to the internet followed by a call to "InternetOpenUrlA" and connects to the site specified in the code "www.maliciousspecimen.com" as illustrated in Figure 32.

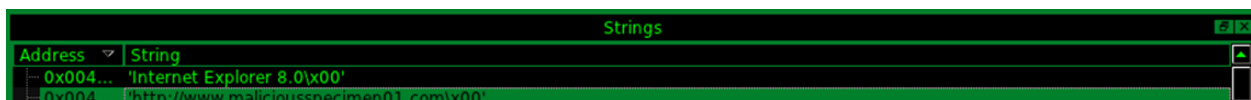


Figure 32: Internet URL connection

5.6 Interactive Behavioural Analysis

Interactive behavioural analysis requires the utilisation of tools such as Wireshark a packet capture tool ran on the disposable machine through REMnux. Wireshark is a network protocol analyser that captures, analyses, and filters network traffic. As illustrated in Figure 33 the malicious PE file under execution attempts to connect to `www.maliciouspecemen01.com`.

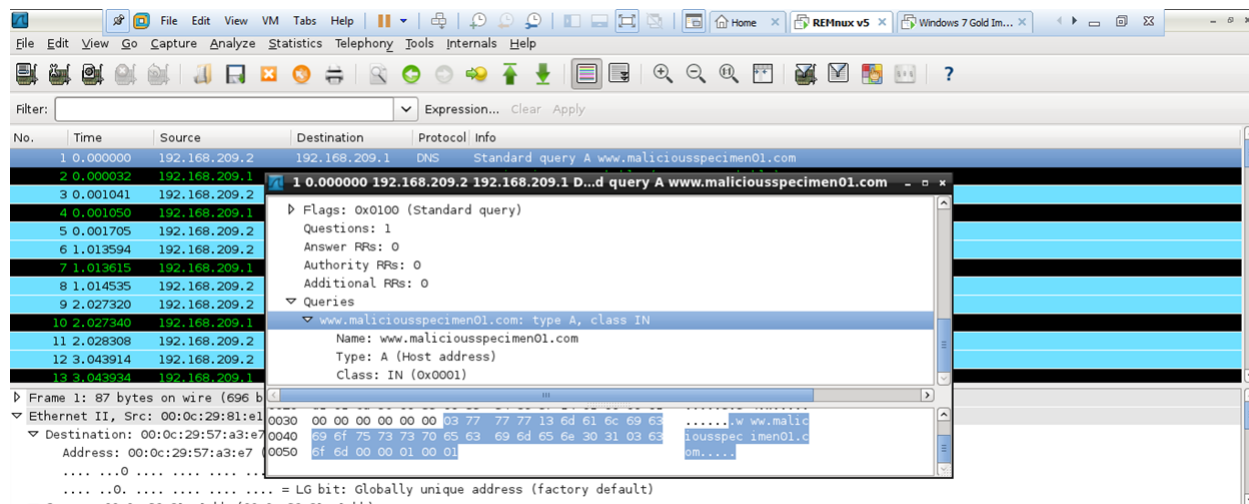


Figure 33: Wireshark Capture from REMnux

5.7 Malware Mitigation Strategy

The concept of a machine getting compromised also referred to as “owned” isn’t a question of “if”, rather a question of “when”, mitigating against the risk that malware compromise a machine requires the generation of a signature that identifies that exact variant prior to infection.

5.7.1 Signature creation in REMnux

As illustrated in Figure 34 using the “sigtool” it is possible to create a MD5 hash based signatures that will uniquely identify the malware specimen as was the intent of this thesis was to analysis a malicious specimen and assess that specimen’s true intent and as a consequence create a signature that will facilitate future detection as per Figure 35: MaliciousSpecimen01.hdb signature validation test.

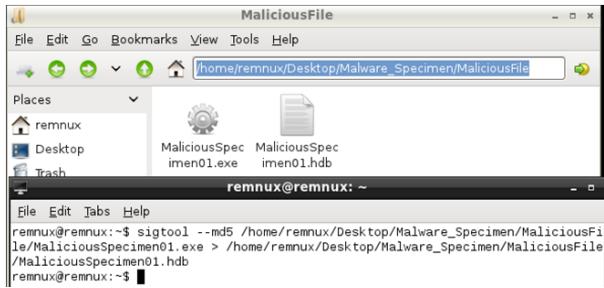


Figure 34: MD5 hash based signature for MaliciousSpecimen01.exe

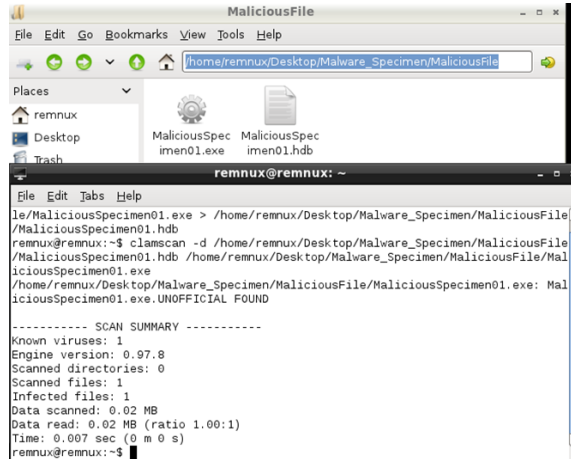


Figure 35: MaliciousSpecimen01.hdb signature validation test

5.7.2 Signature Transferred to Windows 7 Machine

As illustrated below in Figure 36 the “MaliciousSpecimen01.hdb” signature created above has to be transferred to “C:\ProgramData\clamwin\db” where ClamWin’s database is located so that testing of the signature can happen within the windows 7 machine.

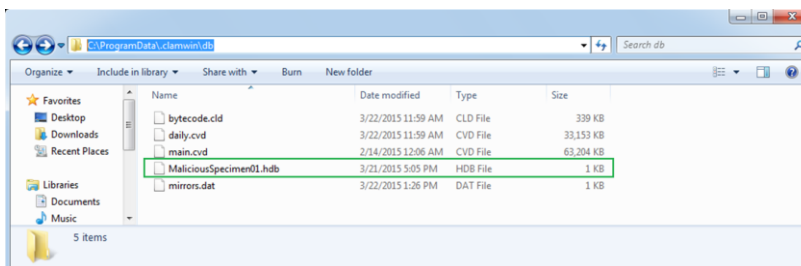


Figure 36: Signature Transferred to Windows 7

5.7.3 Signature Validation on Windows 7 Machine via Command Line

Figure 37 illustrates a scan ran from command line to validation signature functioning correctly with the virus signature database prior to using the GUI.

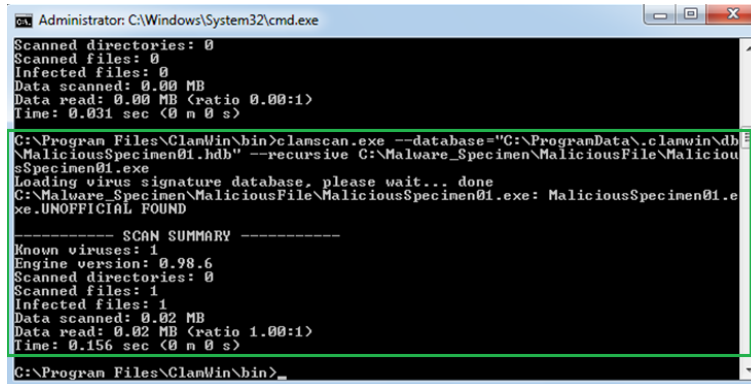


Figure 37: Signature Validation on Windows 7 Machine

The above test verifies that the signature located in the ClamWin database works as expected, identifying the malicious specimen as a threat, however this signature has a status of being “UNOFFICIAL FOUND” authorised vendors would have to secondly validate the true intention of the PE file under analysis.

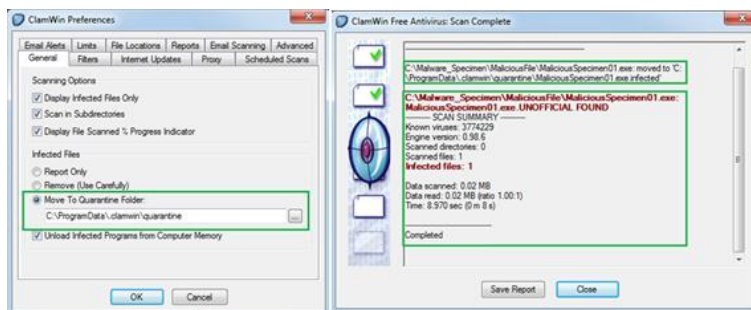


Figure 38: ClamWin AV Scan

On re-scanning the malicious executable is moved to quarantine folder as identified in the above illustration, thus moving an infected file to a specific zone where it cannot initiate further damage within the system as illustrated in Figure 39 moves the “MaliciousSpecimen01.exe” corrupt file is moved to quarantine to the quarantine folder as illustrated in Figure 37 below.

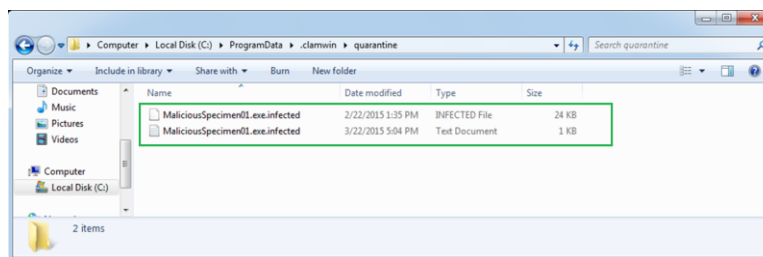


Figure 39: “MaliciousSpecimen01.exe” moved to quarantine

5.7.4 Malware Removal Validation on Windows 7 Machine

On rescan of the affected directory the malicious file no longer exists.

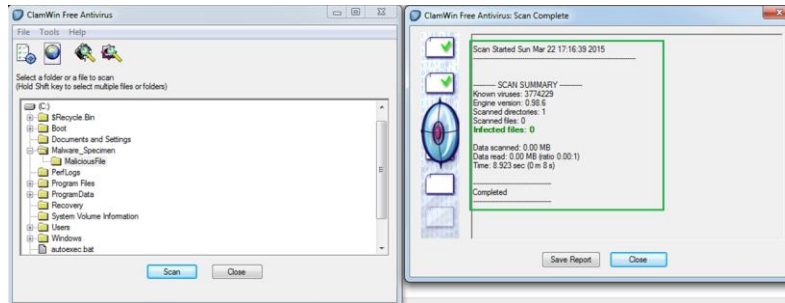


Figure 40: Rescan to validate " MaliciousSpecimen01.exe" removed

5.8 Summary of Findings

As demonstrated in section 5.3 Specimen Selection & Testing, traditional antivirus solution are failing the end user, this is validated by Figures 14 and 15 as the malicious executable "MaliciousSpecimen01.exe" evades detection. Additionally section 5.4 Static Analysis, illustrate that taken a fingerprint of the malicious PE file for utilisation on VirusTotal, an online detection systems verified that the specimen is unknown which is concerning as applying obfuscation techniques such as packing as identified in Figure 20 can result in detection evasion, therefore, online detection systems and anti-virus programs scan for malware occurrences that are already known (Kolbitsch, et al., 2009). Furthermore as illustrated in Figure 19 the malicious specimen wasn't identified by FireEye a commercially available detection systems thus, indicating that the legitimate bona fide detection systems are restricted to recognize only known patterns of malware. Also concerning is the fact that the adoption of packing to obfuscate the specimen resulted in concealed import functions used within Windows OS family for PE files to operate correctly.

Section 5.5 Dynamic Analysis demonstrates the practice of utilising a disassembler to examine the imported functions to identify activity such as imported services and Windows time structure manipulation within the host system. Section 5.6 Interactive Behavioural Analysis illustrates as per Figure 33 the malicious PE file under execution attempting to connect to "www.maliciousspecemen01.com".

Section 5.7 Malware Mitigation Strategy demonstrates the process necessary to mitigate against a specific malicious specimen compromising a machine through the generation of a signature that identifies that exact variant thereby eliminating the risk of infection. Therefore, unknown malware, requires analysts to determine behaviour by analysing PE files code using static, dynamic and interactive behaviour analysis to ascertain the malicious intent within the PE file thus facilitating signature creation and detection systems update to detect future infections. On running Clam AV with the appropriate signature the file was identified as malicious and subsequently placed in quarantine.

6 CONCLUSION

This thesis set out to conduct, evaluate and analyse Windows malware specimen/artefacts in a controlled virtual environment to determine the purpose and function of the specimen/artefact. Both static and dynamic code analysis were used and analysed using WireShark and REMnux thus exposing the malware specimen. Literature review suggested that a multitude of methods exist for the hacker to use in any attack. From the exploitation of known software flaws, exposure of hidden functionality and social engineering, malicious code development is fast becoming a growth criminal activity. While static analysis remains the principal technique for client based malware detection (Christodorescu & Jha, 2004; Griffin, et al., 2009), contemporary malware dynamically detects, identifies and attempts to isolate malware before it can reach a system or network. This has been hampered by progressive, sophisticated development programs designed to evade not only detection but also removal (Konstantinou & Wolthusen, 2008; Balakrishnan & Schulze, 2005). The problem lies in the quality of a malware detector which has to consider the variety of threats that malwares pose. While there are databases and classification schemes which aid in this process, nevertheless an escalation in the detection failure rate of virus/malware detection is still being reported (Panda Security, 2013). Signatures are used by anti-malware detectors to uniquely identify malicious programs. Signature-based anti-malware programs rely upon these pre-defined signatures and heuristic methods (Shafiq, et al., 2009) however the value of this is limited also since malware writers modify each other's development techniques and code fragments. Early research from Christodorescu & Jha, (2004) showed that commercial detection engines are primarily signature-based. This is problematic since modifications of binary code would likely render an original signature ineffective leading to a significant decrease in the detection rate of the original binaries. Termed 'obfuscation' these successive versions are only slightly different from its predecessor making their signature difficult to detect (Shafiq, et al. 2009). The obfuscation strategy employed by malware writers force malware detector writers to use improved detection techniques with each successive release. The ability to 'morph' viruses is to encrypt the malicious payload and decrypt it during execution is a particularly dangerous obfuscation technique. For example, while polymorphic viruses change themselves to evade detection, the more complex metamorphic viruses can also evade heuristic detection algorithms. Security must be understood as a process rather than an endpoint in itself, including the policies, procedures and security awareness training programmes developed both in-house and by independent security firms. This can lead to a more security conscious workforce capable of intelligently dealing with the possibilities of malware detection and removal.

There are significant complexities involved in analysing malware which are exasperated by the ease with which detection evasion may be achieved with minor modifications to the code base of the malware. Beginning with static analysis, code is examined while inert and typically in a

sandbox environment, thereby mitigating the risk of system damage. Dynamic analysis requires that the malware is executed to observe its interaction within a host which validate the findings of static analysis if malware is suspected. This may be supported by host based and online scanning which can detect previously identified malware (see Figure 18, page 33) thus prompting the analyst to conduct further investigations. Unfortunately, the widespread use of packer's which modify an executable file to obfuscate the specimens present in a malware specimen, increase the level of complexity required in analysis. Overcoming aspects of this, behavioural analysis offers the analyst a more sophisticated suite of tools and techniques which can detect and report on the possibility that an executable is obscuring the presence of a packer. This is indicated by a measurement of entropy where high entropy indicates the use of a packer (see Figure 24, page 36).

A number of relevant contributions to both security education and practitioner's context are as follows.

- a) Antivirus on endpoints is not enough, nor is the latest version of antivirus software installed.
- b) Comprehensive endpoint security products which include the following layers of protection will help,
- c) Unpatched vulnerabilities are dangerous, endpoint intrusion prevention systems are needed that protect against this as well as malware attacks that may reach endpoints
- d) The browser is the window to the external world. This needs to be protected against obfuscated web-based attacks from watering holes, spear phishing and social engineering;
- e) Setting application control settings to prevent browser plug-ins from downloading unauthorized and potentially malicious content;
- f) Policies that prevent/limit devices which can be used in an organisation.

However, conducting, evaluating and analysis of the Windows malware specimen/artefacts was not without some limitations.

Limitation of Research

The successes of this research were also limited by several factors such as the time available to complete such a complex study, the expense of software and hardware resources available.

Static analysis has the ability to cover all possible program code paths, even those that usually do not execute, thus yielding a true characterization of that executables entire functionality. This technique is not resource intensive however, time consuming. Static analysis experience run-time packing and many anti-reversing and anti-disassembly techniques including code permutation, encryption, garbage code insertion, code permutation and compression which

was present with the specimen selected and required further study and analysis to understand what packer was utilised.

Dynamic analysis allows for behavioural observation of features including binary obfuscation or run time packers, as changing a malware's binary rarely affects system calls that it invokes. Dynamic analysis may not have complete coverage of an executable's functional compartments therefore, failing to uncover the entire functionalities present in a given malware program. This transpires as monitoring the executed malware dynamic only captures system call traces that correspond directly to the code path during that particular execution. Alternative code paths may be taken during subsequent executions, reliant on the program's internal logic and possibly the program's external environments. A time trigger exists, exhibiting an interesting behaviour only when certain conditions are met, an example may include a bot net that waits for commands from their C2 and malware programs designed to launch attacks at, or before, a certain date and time as is the case with the specimen detail in this thesis. Since their specific conditions are often not met, when executed and monitored in a general environment, these trigger conditioned specimens generate no repeatable run time traces even after modifying the system date and time. Secondly, dynamic analysis is innately resource demanding therefore, limiting coverage.

Further Work

The overarching model which may guide the development and evolution of a security infrastructure in an organisation is the software development lifecycle (SDLC). Its adaptability and consistency are its key strengths but it's important to recognize that the scale and complexity of the problems associated with malware propagation are significant. Today, globally networked computing enables the distribution of malware at an alarming rate making the job of the security professional increasingly necessary and the importance of the tools used to detect, analyse and even predict threats ever more complex. From the end-user's perspective education as regards the importance of security from reputable experts is vital. From the organisational/institutional perspective, a sufficiently trained and security aware workforce helps to protect the reputation of an organisation as a trusted vendor and may lead to an increase in consumer confidence. Supporting this from a technological perspective, defence in depth emphasizes multi layered, supportive security defence systems to protect and organisation. A comprehensive plan to research the best to market solutions as regards antivirus products, intrusion detection systems (IDS), intrusion protection systems (IPS), web application vulnerability scanners is needed as well as investment in security trained teams within that organisation. Future work might focus on the creation of a detection model which would focus on characterising obfuscation techniques thereby distinguishing between malicious behaviour of malware programs rather than identifying specific characteristics for signature creation.

7 REFERENCES

Ahmed, F., Hameed, H., Shafiq, M. Z. & Farooq, M., 2009. *Using spatio-temporal information in API calls with machine learning algorithms for malware detection*. NY, US, ACM, pp. 55 - 62.

AmericanScientific, 2001. *When did the term 'computer virus' arise?*. [Online]
Available at: <http://www.scientificamerican.com/article/when-did-the-term-compute/>
[Accessed 01 12 2014].

AWPG, 2013. *Mobile Threats and the Underground Marketplace*. [Online]
Available at: http://docs.apwg.org/reports/mobile/APWG_Mobile_Report_v1.9.pdf
[Accessed 02 12 2014].

Balakrishnan, A. & Schulze, C., 2005. *Code Obfuscation Literature Survey*. [Online]
Available at: <http://pages.cs.wisc.edu/~arinib/writeup.pdf>
[Accessed 20 October 2014].

Beaucamps, P., 2007. Advanced Polymorphic Techniques. *International Journal of Computer Science*, 2(3), pp. 194-205.

Boren, Z. D., 2014. *The Independent*. [Online]
Available at: <http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html>
[Accessed 02 12 2014].

Bruschi, D., Martignoni, L. & Monga, M., 2007. Code normalization for self-mutating malware. *IEEE Security & Privacy*, 5(2), pp. 46-54.

Christodorescu, M. & Jha, S., 2004. Testing malware detectors. 29(4):34e44.. *ACM SIGSOFT international symposium on Software testing and analysis*, 29(4), pp. 34-44.

Coogan, K., Lu, G. & Debray, S., 2011. *Deobfuscation of virtualization-obfuscated software: a semantics-based approach*. New York, NY, USA, ACM, pp. 275-284 .

Dwan, B., 2000. The Computer Virus — From There to Here.: An Historical Perspective. *Computer Fraud & Security*, 12(2000), p. 13–16.

FireEye, 2014. *Security Reimagined - An Adaptive Approach To Cyber Threats For The Digital Age*, s.l.: FireEye.

Forensickb, 2013. *Forensickb*. [Online]
Available at: <http://www.forensickb.com/2013/03/file-entropy-explained.html>
[Accessed 03 12 2014].

Forum, W. E., 2014. *The Global Information Technology Report 2014: Rewards and Risks of Big Data*. [Online]

Available at: http://www3.weforum.org/docs/WEF_GlobalInformationTechnology_Report_2014.pdf
[Accessed 20 September 2014].

GDATA, 2014. *The early days - History of malware - A brief history of viruses, worms and Trojans.*
[Online]

Available at: <https://www.gdatasoftware.com/securitylabs/information/history-of-malware>
[Accessed 01 12 2014].

Ghiasi, M., Sami, A. & Salehi, Z., 2012. Dynamic malware detection using registers values set analysis. *Information Security and Cryptology (ISCISC)*, pp. 54 - 59.

Griffin, K., Schneider, S., Hu, X. & Chiueh, T., 2009. *Automatic generation of string signatures for malware detection. In: Recent advances in intrusion detection. Springer; 2009. pp. 101e20..* Verlag Berlin Heidelberg, RAID 2009.

Karbalaiie, F., Sami, A. & Ahmadi, M., 2012. Semantic Malware Detection by Deploying Graph Mining. *IJCSI International Journal of Computer Science Issues*, 9(1), pp. 373 - 379.

Kaspersky, 2013. *Global Corporate IT Security Risks*, s.l.: Kaspersky LABS ZAO.

Kolbitsch, C. et al., 2009. *Effective and Efficient Malware Detection at the End Host.* 18th USENIX Security System Montreal, Canada, USENIX Security System.

Konstantinou, E. & Wolthusen, S., 2008. *Metamorphic Virus: Analysis and Detection*, London: Department of Mathematics, Royal Holloway, University of London Egham, Surrey TW20 0EX, England.

Kornblum, J., 2006. *Identifying almost identical files using context triggered.* [Online]
Available at: <http://dfrws.org/2006/proceedings/12-Kornblum.pdf>
[Accessed 12 01 2015].

Kramer, S. & Bradfield, J. C., 2010. A general definition of malware. *J Comput Virol*, Volume 6, p. 105–114.

Lavasoft, 2014. *Lavasoft - Detecting Polymorphic Malware.* [Online]
Available at: <http://www.lavasoft.com/mylavasoft/securitycenter/whitepapers/detecting-polymorphic-malware>
[Accessed 14 02 2015].

Li, X., Loh, P. & Tan, F., 2011. Mechanisms of Polymorphic and Metamorphic Viruses. *Intelligence and Security Informatics Conference (EISIC)*, pp. 149 - 154.

Lyda, R. & Hamrock, J., 2007. *Malware - Using Entropy Analysis to Find Encrypted and Packed Malware.*
[Online]
Available at:
<http://virii.es/U/Using%20Entropy%20Analysis%20to%20Find%20Encrypted%20and%20Packed%20Mal>

ware.pdf

[Accessed 03 12 2014].

Malin, C. H., Casey, E. & Aquilina, J. M., 2012. *Malware Forensics Field Guide for Windows Systems: Digital Forensics Field Guides*. 1 ed. s.l.:Syngress.

McAfee, 2014. *McAfee Labs Threat Report*. [Online]

Available at: <http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014.pdf>

[Accessed 02 12 2014].

O'Kane, P., Sezer, S. & McLaughlin, K., 2011. Obfuscation: The Hidden Malware. *IEEE Security & Privacy*, 09(05), pp. 41-47.

Rabek, J. C., Khazan, R. I., Lewandowski, S. M. & Cunningham, R. K., 2003. *Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code*. New York, NY, USA, Proceedings of the 2003 ACM workshop on Rapid malware, p. 76 – 82.

Rad, B. B., Masrom, M. & Ibrahim, S., 2011. Evolution of Computer Virus Concealment and Anti-Virus. *IJCSI International Journal of Computer Science Issues*, 9(1), pp. 113 - 121.

Rolles, R., 2009. *Unpacking virtualization obfuscators*. Montreal, Canada, USENIX conference on Offensive technologies.

SANS Institute, 2013. *Secure Coding. Practical steps to defend your web apps.* [Online]

Available at: <http://software-security.sans.org/resources/paper/cissp/defining-understanding-security-software-development-life-cycle>

[Accessed 13 December 2013].

Schiffman, M., 2010. *A Brief History of Malware Obfuscation: Part 1 of 2*. [Online]

Available at: <http://blogs.cisco.com/security>

[Accessed 20 October 2014].

Security, P., 2013. *Annual Report PandaLabs, 2013 summary*. [Online]

Available at: http://press.pandasecurity.com/wp-content/uploads/2010/05/PandaLabs-Annual-Report_2013.pdf

[Accessed 20 October 2014].

Shafiq, M. Z., Tabish, S. M. & Farooq, M., 2009. *PE-Probe: Leveraging Packer Detection and Structural Information to Detect Malicious Portable Executables*. Switzerland, Virus Bulletin Conference(VB).

Solomon, A., 1993. A Brief History of PC Viruses. *Computer Fraud & Security Bulletin*, 1993(12), pp. 9-19.

Song, Y. et al., 2007. *On the infeasibility of modeling polymorphic shellcode*. Alexandria, Virginia, ACM.

Szor, P., 2005. *The Art of Computer Virus Research and Defense*, MA, USA: Addison-Wesley,.

- Ször, P. & Ferrie, P., 2001. *Symantec Hunting For Metamorphic*. [Online]
Available at: <https://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf>
[Accessed 02 12 2014].
- TrendLabs, 2014. *Turning the Tables on Cyber Attacks*, s.l.: TREND MICRO.
- Verizon, 2014. *2014 DATA BREACH INVESTIGATIONS REPORT*, s.l.: Verizon.
- Von Neumann, J., 1951. *Design of Computers Theory of Automata and Numerical Analysis*. [Online]
Available at: <http://www.sns.ias.edu/~tlusty/courses/InfoInBio/Papers/vonNeumann1951.pdf>
[Accessed 01 12 2014].
- Wong, W. & Stamp, M., 2006. Hunting for Metamorphic Engines. *Journal in Computer Virology*, 2(3), pp. 211-229.
- Xufang, L., Loh, P. & Tan, F., 2011. *Mechanisms of Polymorphic and Metamorphic Viruses*. Athens, IEEE, pp. 149 - 154.
- Yason, M. V., 2007. *The art of unpacking*. [Online]
Available at: <https://www.blackhat.com/presentations/bh-usa-07/Yason/Presentation/bh-usa-07-yason.pdf>
[Accessed 27 12 2014].
- You, I. & Yim, K., 2010. *Malware Obfuscation Techniques: A Brief Survey*. Fukuoka, IEEE, pp. 297-300.

8 APPENDICS

Appendix 1: Glossary of Terms

Malwr: An open source malware analysis service and community launched in January 2011. You can submit files to it and receive the results of a complete dynamic analysis back. <https://malwr.com/>

Virus Total: Free service that analyses suspicious files and URLs and facilitates the quick detection of malware such as viruses, worms, Trojans, and etc. <https://www.virustotal.com/>

REMnux v5: <http://digital-forensics.sans.org/blog/2013/04/10/installing-remnux-virtual-appliance>

Mutex - Short for *mutual exclusion object*. In computer programming, a mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously. When a program is started, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. The mutex is set to unlock when the data is no longer needed or the routine is finished.

HTTP Daemon - is a software program that runs in the background of a web server and waits for the incoming server requests. The daemon answers the request automatically and serves the hypertext and multimedia documents over the internet using HTTP. httpd stands for Hypertext Transfer Protocol Daemon.

Comodo Valkyrie - Automated Analysis System of Portable Executable (PE) file's (.exe, .dll, .sys etc). Service available at: <https://valkyrie.comodo.com/>

Appendix 2: FireEye Report

Below is a report generated by FireEye.

Malware Analysis (as of 03/16/15 11:31:02 EDT)

Submit Analysis

Live Sandbox Timeout: Priority: Profile: Force:

URL: Note:

Params:

Cancel Queued Analysis

Filter Analysis

Pattern: Column: Case Sensitive: Submitted By:

Pages: [1](#) [2](#) [3](#) ... [100](#) There are a total of 1858 malware analysis for the current filter. A maximum of 100 pages of results will be displayed. [\[show all events\]](#)

ID	Type	Analysis	Malware	URL	Profile Name - Application	MD5sum	Submitted (EDT)	Complete (EDT)	Status	Submitter
2131	misc	No Sandbox		file://Payroll_pdates_2008.XLS	winxp-sp3 -		03/13/15 18:22:58	03/13/15 18:22:59	File Unknown	admin
2130	misc	No Sandbox		file://Payroll_pdates_2008.XLS	win7x64-sp1 -		03/13/15 18:22:58	03/13/15 18:22:59	File Unknown	admin
2129	misc	No Sandbox		file://Payroll_pdates_2008.XLS	win7-sp1 -		03/13/15 18:22:58	03/13/15 18:22:59	File Unknown	admin
2128	exe	No Live		file://MaliciousSpecimen01.exe	winxp-sp3 - Windows Explorer	4dbbfcd27ec5d555dd0b0e68d20f4ea2	03/13/15 09:35:42	03/13/15 09:44:26	Success (text) [download clip]	admin
2127	exe	No Live		file://MaliciousSpecimen01.exe	win7x64-sp1 - Windows Explorer	4dbbfcd27ec5d555dd0b0e68d20f4ea2	03/13/15 09:35:41	03/13/15 09:44:26	Success (text) [download clip]	admin

Malware: Malware.Binary.exe VM Capture(s): [\[1\] pcap_6432 bytes \(text\) \[download clip\]](#)
 Application Type: Windows Explorer Analysis OS: [\[2\] extracted files_8466576 bytes](#)
 File Type: exe Archived Object: [Microsoft.Windows7_64-bit_6.1.sp1_14.1110](#)
[4dbbfcd27ec5d555dd0b0e68d20f4ea2.zip](#)

Suspicious Behavior Observed

OS Change Detail (version: 1.679) | Items: 32 | OS Info: Microsoft Windows7 64-bit 6.1.sp1 14.1110 [Top](#)

Type	Mode/Class	Details (Path/Message/Protocol/Hostname/Gtype/ListenPort etc.)	Process ID	Parent ID	File Size
Analysis	Malware				
Application					
Eventlogcmd					
Os		Name: windows Version: 6.1.7601 Service Pack: 1 SequenceNumber: 4			
Os Monitor		Build: 284586 Date: Nov 7 2014 Time: 15:57:45 Version: 14R2 Sequenc eNumber: 5			
Eventlogcmd					
Uac	Service	Background Intelligent Transfer Service			
Uac	Service	SSDP Discovery			
Uac	Service	Software Protection			
Uac	Service	Portable Device Enumerator Service			
Uac	Service	Security Center			
Uac	Service	TCP/IP NetBIOS Helper			
Uac	Service	TCP/IP NetBIOS Helper			
Process	Started	C:\Users\Administrator\AppData\Local\Temp\MaliciousSpecimen01.exe Parentname: C:\Windows\SysWow64\cmd.exe Command Line: "C:\Users\ADMINI~1\AppData\Local\Temp\MaliciousSpecimen01.exe" MD5: 4dbbfcd27ec5d555dd0b0e68d20f4ea2 SHA1: 70031437c885e9068b50d6a6f8968d8d7db438	496	2628	24576
Malicious Alert	Generic Process Launch Activit y	Message: Startup behavior anomalies observed Detail: A new process has been launched			
File	Failed	C:\Windows\System32\Wow64\LOG.DLL	496		
File	Failed	C:\Users\ADMINI~1\AppData\Local\Temp\VERSION.DLL	496		
Regkey	Queryvalue	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Control\ComputerName\ActiveComputerName\Com puterName*	496		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\BITS*Start* = 0x00000003	484		
API Call		API Name: StartServiceCtrlDispatcherA Address: 0x0040102e	496		

API Call		API Name: StartServiceCtrlDispatcherA Address: 0x0040102e Params: [0x18f3c] ImagePath: C:\Users\Administrator\AppData\Local\Temp\MaliciousSpecimen01.exe D LL Name: advapi32.dll	496		
Mutex		\Sessions1\BaseNamedObjects\HGL345	496		

API Call		API Name: CreateServiceA Address: 0x004010c2 Params: [0x4e5b58, Malservice, Malservice, 2, 16, 2, 0, C:\Users\ADMINI-1\AppData\Local\Temp\MaliciousSpecimen01.exe, NULL, 0x0, NULL, NULL, NULL] ImagePath: C:\Users\Administrator\AppData\Local\Temp\MaliciousSpecimen01.exe D LL Name: advapi32.dll	496		
Regkey	Added	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice\Type = 0x00000010	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice\Start = 0x00000002	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice>ErrorControl = 0x00000000	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice\ImagePath = C:\Users\ADMINI-1\AppData\Local\Temp\MaliciousSpecimen01.exe	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice\DisplayName = Malservice	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice*ImagePath = 0x00000001	484		
Regkey	Setval	\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\Malservice*ObjectName = LocalSystem	484		
API Call		API Name: SystemTimeToFileTime Address: 0x004010eb Params: [0x18fb30, 0x18fb40] ImagePath: C:\Users\Administrator\AppData\Local\Temp\MaliciousSpecimen01.exe D LL Name: kernel32.dll	496		

Click on Tools and Comment to create, send and mark-up PDF files.

Additional Information: [Show all](#)

Tool Name and Version	Tool Output
Name: fe_sigcheck Version: 0.9	Tool Output: Authenticode Signature Unsigned binary

Name: fe_peinfo Version: 0.9	Tool Output: PE file info [[Basic Info]] EntryPoint Address : 0x1190 Image Base : 0x400000 TimeStamp : 0x4e861d38 (Fri Sep 30 15:49:12 2011) MachineType : 0x14c [[File Info]] No File Info present. [[3 Section(s)]] Name vrtaddr vrtsize rawsize md5 sha1 .text 0x00001000 0x0000295E 0x00003000 664ad4360283dd1937af16c76b585511 c822c7a97384d0e85b1efccdd7b3eee9dd66038 .rdata 0x00004000 0x000008CA 0x00001000 5c0b117eb9379a1e2ed40e963cb2d1a9 42c77328d2a22a161fbc8d0104f5edaac11023c7f .data 0x00005000 0x000007FC 0x00001000 a24dc721d4bf0822d034a906e17e0629 d0a686822e327226461456d34c0ac85668e9464b [[3 Import(s)]] KERNEL32.dll 0x404010 CreateWaitableTimerA 0x404014 SystemTimeToFileTime 0x404018 GetModuleFileNameA 0x40401c SetWaitableTimer 0x404020 CreateMutexA 0x404024 ExitProcess 0x404028 OpenMutexA 0x40402c WaitForSingleObject 0x404030 CreateThread 0x404034 GetCurrentProcess 0x404038 Sleep 0x40403c GetStringTypeA 0x404040 LCMapStringW 0x404044 LCMapStringA 0x404048 GetCommandLineA 0x40404c GetVersion 0x404050 TerminateProcess 0x404054 UnhandledExceptionFilter 0x404058 FreeEnvironmentStringsA 0x40405c FreeEnvironmentStringsW 0x404060 WideCharToMultiByte 0x404064 GetEnvironmentStrings 0x404068 GetEnvironmentStringsW 0x40406c SetHandleCount 0x404070 GetStdHandle 0x404074 GetFileType 0x404078 GetStartupInfoA 0x40407c HeapDestroy 0x404080 HeapCreate 0x404084 VirtualFree 0x404088 HeapFree 0x40408c RtlUnwind 0x404090 WriteFile 0x404094 HeapAlloc 0x404098 GetCPInfo 0x40409c GetACP 0x4040a0 GetOEMCP 0x4040a4 VirtualAlloc 0x4040a8 HeapReAlloc 0x4040ac GetProcAddress 0x4040b0 LoadLibraryA 0x4040b4 MultiByteToWideChar 0x4040b8 GetStringTypeW ADVAPI32.dll
---------------------------------	---

Name	Version	Tool Output
exiftool	8.50	<pre> ExifTool Version Number : 9.27 File Name : 2127.malware Directory : /data/malware/gdone File Size : 24 kB File Modification Date/Time : 2015:03:13 09:35:40+04:00 File Access Date/Time : 2015:03:13 09:44:26+04:00 File Inode Change Date/Time : 2015:03:13 09:44:26+04:00 File Permissions : rw-r--r-- File Type : Win32 EXE MIME Type : application/octet-stream Machine Type : Intel 386 or later, and compatibles Time Stamp : 2011:09:30 15:49:12+04:00 PE Type : PE32 Linker Version : 6.0 </pre>
2126	exe	<pre> No Sandbox File: file:///MaliciousSpecimen01.exe Initialed Data Size: 8192 Uninitialized Data Size: 0 Entry Point: 0x1190 </pre>
2125	exe	<pre> No Sandbox File: file:///MaliciousSpecimen01.exe Image Version: 0.0 Subsystem Version: 4.0 </pre>
2124	exe	<pre> No Sandbox File: file:///MaliciousSpecimen01.exe </pre>
2123	exe	<pre> No Sandbox File: file:///MaliciousSpecimen01.exe </pre>
2122	pdf	<pre> No Sandbox File: file:///WHISTLEB*CK-LAKE-INC.pdf </pre>

Page: 1 2 3 ... 168 [XML dump](#)

Appendix 3: REMnux Machine Configuration

The necessary commands to perform steps 3 to 6 are:

Step 3

```
remnux@remnux:~$ sudo fakedns 192.168.209.23
```

```

remnux@remnux: ~
File Edit Tabs Help
remnux@remnux:~$ sudo fakedns 192.168.209.23
pyminifakeDNS:: dom.query. 60 IN A 192.168.209.23

```

Step 4

Open another shell:

```
remnux@remnux:~$ httpd start
```

```

remnux@remnux: ~
File Edit Tabs Help
remnux@remnux:~$ httpd start
Starting web server: thttpd.

```

Step 5

```
remnux@remnux:~$ sudo sysctl -w net.ipv4.ip_forward=1
```

```

remnux@remnux:~$ sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1

```

Step 6

```
remnux@remnux:~$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 808
```

```
remnux@remnux:~$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j REDIRECT --to-port 8080
```

```
remnux@remnux:~$ sudo iptables -t nat -L
```

```
Chain PREROUTING (policy ACCEPT)
```

target	prot	opt	source	destination
REDIRECT	tcp	--	anywhere	anywhere tcp dpt:https redir ports
REDIRECT	tcp	--	anywhere	anywhere tcp dpt:www redir ports
REDIRECT	tcp	--	anywhere	anywhere tcp dpt:www redir ports
REDIRECT	tcp	--	anywhere	anywhere tcp dpt:https redir ports

```
remnux@remnux:~$ burpsuite
```

```
[1] 8912
```

```
remnux@remnux:~$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
remnux@remnux:~$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j REDIRECT --to-port 8080
remnux@remnux:~$ sudo iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target    prot opt source                destination
REDIRECT  tcp  --  anywhere              anywhere          tcp dpt:www redir ports 8080
REDIRECT  tcp  --  anywhere              anywhere          tcp dpt:https redir ports 8080

Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination

Chain POSTROUTING (policy ACCEPT)
target    prot opt source                destination
remnux@remnux:~$ burpsuite
```