An analysis of malware evasion techniques against modern AV engines

Submitted in partial fulfillment
of the requirements of the degree of

Master of Science

of Rhodes University

Jameel Haffejee

Grahamstown, South Africa
July 11, 2015

Abstract

This research empirically tested the response of antivirus applications to binaries that use virus-like evasion techniques. In order to achieve this, a number of binaries are processed using a number of evasion methods and are then deployed against several antivirus engines. The research also documents the process of setting up an environment for testing antivirus engines, including building the evasion techniques used in the tests. The results of the empirical tests illustrate that an attacker can evade multiple antivirus engines without much effort using well-known evasion techniques. Furthermore, some antivirus engines may respond to the occurrence of an evasion technique instead of the presence of any malicious code. In practical terms, this shows that while antivirus applications are useful for protecting against known threats, their effectiveness against unknown or modified threats is limited.

Acknowledgments

I would like to thank everyone who has helped and supported me during the researching and writing of this thesis. To my parents and Haroon Meer, thank you for encouraging me to get started. My supervisors Barry Irwin, Yusuf Motara and Adam Schoeman: I am grateful for all the constant and invaluable feedback that you provided and for guiding me through the entire process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Antivirus software has become one of the largest commercial industries in regards to computer security with an estimated 50 antivirus products competing to protect the end user. This boom has been born largely out of the virus arms race which began in the late 1980s (Moore *et al.*, 2009, Hsu *et al.*, 2012, VirusTotal, 2014). During this period, virus authors and antivirus companies fought aggressively: the malware authors fought to gain as large an infection base as possible while the antivirus companies fought to remove these viruses (once they were infected) or to prevent them from infecting the end user's system in the first place. In response malware authors actively started preempting antivirus companies detecting their malware by using scanning services (Krebs, 2009). While antivirus companies have become successful at cleaning infected systems, the shift has moved toward prevention rather than cure (Moore *et al.*, 2009). This change from curing a system of its infection to preventing the infection in the first place stems from the complexity involved with current malware. Even after successful removal it can't be guaranteed that the system is completely disinfected and there always remains the possibility that the malware modified the system in a way the antivirus company was not aware off.

The reasoning behind this (being that since the antivirus companies do not have access to source code for a piece of malware) it cannot accurately say that it has cleaned an end user's system to the extent that the system is in the same clean state it was before the attack. Furthermore, once a piece of malware is able to get access to an end user's system, it is not beyond their capability to disable an antivirus engine without alerting the end user (Alsagoff, 2008). This, in turn, renders the antivirus product ineffective at protecting against further future threats.

To get around the protection that the antivirus engines developed to protect end users,

malware authors changed the way the malware was packaged and distributed. In response to the subsequent modifications by antivirus authors, a number of evasion techniques were evolved until a new evasion technique was developed. The techniques employed by malware authors during the early 1990s are still in use today and will be discussed in further detail in section 2.5. While these techniques have remained fairly unchanged, very little formal research has been completed in the area of antivirus evasion analysis. The research presented in this work aims to test these techniques on modern systems and evaluate their effectiveness against modern antivirus engines.

1.2 Research Question

The purpose of the research documented in this thesis is to investigate whether binaries which exhibit known virus-like evasion techniques can prove to be effective against modern antivirus engines using on-demand scanning techniques. Furthermore, the research attempts to determine if the antivirus applications react to the presence of the evasion technique instead of the malicious code embedded within. Chapters 5, 6 and 7 are dedicated to exploring the different methods of testing the evasion techniques. The effectiveness will be evaluated on a per chapter basis and goals will be defined as to what is expected from the technique being evaluated. Each chapter will also undertake to identify if the evasion technique targeted or the malicious code is being detected by the antivirus engines.

1.2.1 Hypothesis

It is expected that the evasion techniques being tested as well as the binaries being tested will exhibit certain signatures that antivirus engines have already recorded. These signatures are what will be detected and once modified by an evasion technique will allow the binary being tested to pass undetected by the antivirus engines. This will be demonstrated by using known clean binaries and applying an evasion technique. It is expected that once applied, the antivirus engines will detect the benign binary as malicious. This would indicate that the antivirus engines are detecting the evasion techniques based on static signatures instead of the actions performed by the evasion technique.

1.3 Limitations

One of the limitations of performing research into older malware, and the techniques used to evade antivirus engines, is the absence of academic research in this area. As such, many of the sources are taken directly from the 'Virus Exchange (VX) Scene' and the

work that was published by these groups (Herm1t, 2002). The VX Scene is the term applied to various groups that participated in the art of writing malware. These groups would share information between each other via Electronic Text Magazines called E-Zines (Knight, 2005). It is from here, that a fair amount of information will be referenced, as this is considered the source from which virus authors got their ideas and shared findings and techniques on building the latest malware (Thompson, 2004). A further limitation is not being able to test polymorphic and more advanced evasion techniques that are more dynamic. It is hoped that further research performed in this area will be able to focus in on this area as it has significant potential to advance the field of antivirus evasion. The research will focus primarily on the encryption and packing techniques - as they are the only non-dynamic and evolving techniques. The last mentioned limitation is that of constrained resources (of both time and money) required to explore the area of building a sophisticated, custom malware lab and the automation of this process for future research. These constraints will be discussed in section 4. Further the tests will only be performed on windows based malware and does cover testing mobile or linux based malware.

1.4 Conventions

URL Referencing: During the course of the document there are instances in which a URL link needed to be included. Due to the length of certain URLs they will be included in the appendix and cross referenced via the footnotes on the relevant page.

Text Highlighting: Throughout the rest of the document commands that needs to be executed will be emphasized using italics and where required the output will be displayed in bold text. Output that is greater than a single line of 80 characters will be put into tabular form.

Hashes: Throughout the rest of the document binaries will often be listed by the output of a hash function against the binary. This is performed as concrete means of referencing a binary. If for example the binary is renamed while its contents remain the same, the cryptographic hash will remain the same. The hashing function used against the binaries in the rest of this document is the SHA256 function, which provides a sufficient level of entropy that a collision will not occur when hashing two binaries. To save space in the rest of the document the results from the hashing will be reduced from 64 characters to 12 characters, this reduction will be indicated by ellipses. When performing these reductions, care was taken to ensure that there are no duplicate hashes which could result in confusion.

1.4.1 Malware Classification

There are a number of terms that need to be explained in detail regarding the classification of malware. In general this document will refer to trojans, viruses or worms when referring to malware. These terms will be explained in further detail in section 2.2 under malware classification.

1.5 Document Structure

The remainder of the document is structured as follows:

Chapter 2 covers the categories of evasion techniques that have been used in the past as well as those that are still currently in use. The chapter will begin with a high level overview of attributes that define each category. The rest of the chapter will further detail the evasion techniques. Some details covered include when the technique was first detected and if the evasion technique is an evolution of a previous technique, or a new technique. The chapter deals with each of the scanning techniques, which will usually relate directly to an evasion technique used by malware. This will allow us to identify how the viruses evolved and how the techniques that were implemented drove the countermeasures antivirus engines used for scanning. We will observe that certain techniques will fall away and are replaced by newer methods.

Chapter 3 discusses the building of the laboratory that will be used to automate and streamline the process of testing antivirus applications. This also covers how the binaries will be submitted as well as how the results are stored.

Chapter 4 explains how the antivirus engines will be tested by using a Black Box approach of submitting binaries which have a known state and then monitoring the results returned by the antivirus applications.

Chapter 5 focuses on the testing of the packer evasion type and its effectiveness against antivirus engines.

Chapter 6 covers the testing of the encryption evasion type and its effectiveness against antivirus engines.

Chapter 7 combines the efforts from the two previous chapters to determine if combining these evasion methods can result in a greater reduction of antivirus detection rates.

Chapter 8 presents the results from the preceding three core chapters as well as a proposal for future work.

Chapter 2

Literature Review

2.1 Introduction

This chapter begins by introducing some common terms that will be used in the rest of the document in section 2.2. After covering the basic terms used in the document, the chapter covers defensive measures used by malware to protect themselves from being analysed in section 2.4. The chapter then documents the evolution of the known evasion techniques used by malware authors in subsections 2.6 - 2.10. This is followed up by discussing the scanning countermeasures implemented by antivirus applications over time in section 2.11. Once the work on antivirus scanning evolution has been presented, related work will be introduced in section 2.12. Thereafter, the chapter will be brought to a close with the conclusion and a brief overview of the work in the next chapter. The following section begins by dealing with a number of terms which were introduced in chapter 1 and will be explained in detail.

2.2 Malware Terms

The term malware (short for **Mal**icious Soft**ware**) covers a broad spectrum of malicious applications that can be found on a user's system (Lin, 2008). As such, the research will focus on the evasion methods used by three specific types of malware, namely: viruses, trojans, and worms. These are only a few of the terms that are used to classify malicious binaries, but are the terms that relate to the current work set. Before getting into the details for each of the terms, it is useful to define what is considered a malicious binary. A malicious binary is generally defined as any binary application that can affect the end user system in unintended manner without the user being aware (Kramer and Bradfield, 2010). The ignorance of the user, as well as the transformation of the system, is an important

distinction. System administration tools such as Netcat[1], Nmap[2] and PSExec[3] exist to manage and secure the systems they manage. Even though these tools but are not built with a malicious intent, they are often used by attackers to compromise an end user's system (Little, 2005). McAfee (2005) describes these tools as being potentially unwanted programs (PUP) by an antivirus engine to indicate they may cause harm to an end user's system.

2.2.1 Trojan

The trojan is a type of non-viral malicious application that takes its name from Greek mythology (Gordon and Chess, 1998). According to the myth, the Greek Army cleverly gained access to the trojan city, disguised as an innocuous wooden horse (Gordon and Chess, 1999, Burgess, 2003). The key feature of a trojan horse application is that superficially it claims to be a safe or benign application to an end user, but secretly provides a number of malicious actions to the person or group controlling the trojan horse application. A trojan is commonly used to deliver keyloggers or a remote administration tool which would allow the person controlling the trojan to steal data or gain full control over the end user's system (Gordon and Chess, 1999). A trojan does not attempt to spread on its own, instead it masquerades as a harmless application (Gordon and Chess, 1998). By analysing existing trojans, Zolkipli and Jantan (2010) were able to extract six sub-classifications for trojans. These sub-classifications are:

- Packed trojan
- Dropper trojan
- Downloader trojan
- Clicker trojan
- Gamethief trojan
- Backdoors trojan

For the purpose of this research, the dropper trojan is relevant and will be extrapolated. The term "dropper malware" is often used as a synonym for trojans because there is very little distinction between the two. The dropper classification refers to a trojan that exists to deliver other malware in the form of keyloggers or system backdoors (Zolkipli and

[1] `http://netcat.sourceforge.net/`

[2] `http://nmap.org/`

[3] `http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx`

Jantan, 2010). The dropper trojan is significant to the current research because it is often used in a multistage form of attack (Funk and Garnaeva, 2013). This multi-stage attack pattern allows the malware author to gain a foothold on the targeted system and then escalate to full control (Funk and Garnaeva, 2013). Ramilli and Bishop (2010) say that the multi-stage attack pattern also allows for the first stage of the attack to be hidden from antivirus engines much more easily than the second stage attack owing to its smaller payload.

The term "dropper", as applied to malware, has recently increased in usage because of the term being applied to the binary that is distributed in watering hole attacks (Funk and Garnaeva, 2013). A watering hole attack is an attack in which a legitimate website is compromised and then used to deliver malicious trojans to a target's system (Doherty and Gegeny, 2013). The binary that is dropped to the targets computer is very small and meant to establish a foothold on the targets computer.

2.2.2 Virus

A virus is usually characterised by a small application that is designed to spread by injecting malicious code into other binaries (Cohen, 1987). The resultant effect on the end user's system is meant to interfere with normal operations. Ször (2005) explains that viruses tend to affect binary applications and replicate by modifying other binaries. This is in contrast to worms which replicate as a whole and target systems instead of applications.

The main purpose of a virus is to replicate and cause as much damage as possible to the end user's system or spread a message that the author wished to distribute (Sanok, 2005). Viruses are not as popular today as they were during the early 2000s and as such, not encountered nearly as much, this is evident from analysing the malware threat reports between 2001 and 2014 (Global Research Analysis Team, 2001, Garnaeva *et al.*, 2014) and observing the increase in trojans in the most popular malware threats sections, while observing a decrease in viruses decreasing in rank in the most popular threats between 2001 and 2014. While viruses are not as popular as trojans, they did popularize many of the evasion techniques that will be covered in chapter 2.

The lifetime or generation for a piece of malware varies depending on the malware being discussed as well as the type of malware. In respect to simple viruses, the lifetime for a virus refers to how long it was out in the wild and until it was cleaned out by antivirus software (Kephart and White, 1991). The generation of a virus generally refers to the version of virus. Each new version of the virus would be considered a new generation. When covering the more advanced malware in the later sections on polymorphic and metamorphic malware, the lifetime refers to how long a virus existed in a state in which

a single signature could detect that instance of the virus. The generation still refers to the versions of a virus, where a new version would relate to a new feature being added.

2.2.3 Worm

Weaver *et al.* (2003) describes a worm as a self-propagating program that attempts to spread by exploiting vulnerabilities in a computer system (Anderson, 1972). This ability to self-propagate is what makes a worm particularly dangerous. The danger that self-propagation poses can also be construed as a weakness of this class of malicious application. If the systems that the worm targets are patched such that the vulnerability that the worm exploited no longer exists, the worm will be unable to spread (Staniford *et al.*, 2002). The countermeasure to stop worms spreading between systems is simply patching a system as soon as a vulnerability is reported (Kienzle and Elder, 2003, Yu *et al.*, 2010). Rescorla (2003) demonstrates that even when companies are aware of critical issues, there is usually a significant lag in the time it takes to patch these systems. This lag time and inefficiency from companies is what worm authors depend on in order to gain as large an infection base as possible.

2.3 Malware Defence

With the terms in section 2.2 defined, the following sections provide an explanation for each of the techniques that malware authors have employed in the past to defend their creations. The techniques are sequenced historically from oldest to most recent.

2.4 Code Armouring

Code Armouring is the process of obfuscating and/or altering code in order to prevent it from being analysed. While armouring itself is not, by design, specifically used to evade antivirus engines, it is used to prevent automated and static analysis of the code, which will result in a prolonged lifetime for the malware (Sikorski and Honig, 2012, Quist and Smith, 2007).

Before getting into the specifics about the techniques malware authors employed to evade antivirus applications, it is worth noting that most malware authors will try to harden their malware through the use of code armouring to prevent analysis. The techniques used by malware authors to prevent analysis will differ from author to author, though most will use a series of obfuscations and traps to prevent debugging and disassembly (Quist and Smith, 2007). Listed below are areas identified for which code armouring tries to provide protection (Ször, 2005):

- Anti-disassembly
- Anti-debugging
- Anti-emulation
- Anti-VirtualMachine
- Anti-goat

These areas will be briefly explored to provide a deeper understanding as to the reasoning behind their usage by malware authors. The techniques are dealt with in the same order that a malware analyst would typically use to analyse a piece of malware (Harper *et al.*, 2011). Step one: analysing the application in an offline or dormant state. Step two: moving to an online state, where the application is debugged. Step three, the final phase: is usually to leave the application in a running state, in order to collect information about application behaviour the first two steps may have missed.

2.4.1 Anti-disassembly

Anti-disassembly is the process of protecting an application from being disassembled to determine the internal workings of the application as (Aycock *et al.*, 2006). Sikorski and Honig (2012) describe the process of anti-disassembly as being commonly achieved through the use of two groups of techniques. The first group of techniques involves code obfuscation through dynamic code. This attempts to hide the intent of the code by dynamically changing the code as it runs. The code could involve incrementing the data at a specific address by a known value and then jumping to this location after it is modified. While being analyzed by a disassembler, the code that is modified will simply look like junk instructions to the application as it has not yet been changed. This method of obfuscation is what was used by early viruses that encrypted themselves, such as the Cascade virus. The second group of techniques largely revolves around attacking different disassemblers through code and data in the application. By causing the code in the application to be deliberately misinterpreted, the malware author can cause the disassembler to display an incorrect view of what the application is doing internally.

2.4.2 Anti-debugging

While anti-disassembly protects an application from static and offline analysis, anti-debugging is meant to protect an application while it is executing a task (Branco *et al.*, 2012). The techniques implemented are too numerous to detail here, but are described in great detail by Branco *et al.* (2012, Section 3). In general, anti-debugging techniques revolve around

attempting to set the application into a debug state prior to the analysts debugger being attached. Since an application can only be debugged by one debugger at a time, this generally prevents any run-time analysis from succeeding. The alternative approach is to check via a number of a system calls which are dependent on the operating system to determine if a debugger is attached to the current process and then exit if one is detected. The exact details of how these techniques are implemented will vary between operating systems and hardware implementations.

2.4.3 Anti-emulation

In conjunction with anti-debugging techniques, malware authors will often implement anti-emulation techniques in their malware. This is to prevent a malware analyst from gaining a deeper understanding of the malware when executed on an emulated system (Konstantinou and Wolthusen, 2008). The use of emulation provides a malware analyst with the ability to trace the inner workings while the application is running and to substitute instructions if an instruction is not available for the hardware where the malware is being analysed. Emulation is a key technique in preventing malware from running successfully. By emulating the environment of the target system, the scanning application can reasonably determine if the actions of the code are malicious. After second generation (introduced later in section 2.11.2) scanners were introduced, emulation played a significant role in the detection of malware. It is in response to this, that anti-emulation played such a key point in evading antivirus applications. It should be noted that most malware that implemented anti-emulation in their code, did so to remain undetected by the process of not executing their malicious code. While this is important, is it not a generic enough technique that can be applied to a compiled application and requires access to the source code of the application.

2.4.4 Anti-Virtual Machine

The anti-vm (anti-virtual machine) defences for malware are very similar to those of the anti-emulation defence and exist to prevent malware from being analyzed and tested as explained (Ször, 2005). With virtual machine technology becoming more common thanks to free applications e.g. QEMU and Virtual Box, more analysts are able to test malware in the context of a virtual machine. This presents a major and unwanted problem to malware authors, as the environment is almost identical in every manner to a non-virtual machine. The means employed to detect virtual machines are similar to those used to detect debuggers (Dinaburg *et al.*, 2008), in that they either check for a known variable or attempt to access a restricted piece of hardware that would only be available in a virtual

machine (Rin, 2013).

2.4.5 Anti-Goat

Anti-goat defences are currently non-existent since the countermeasure they attempted to defend against is no longer used. When viruses were more common and file infection was a common problem, antivirus companies would routinely create "sacrificial goat" type files that would be purposefully infected (Ször, 2005, Aycock, 2006). Once infected the goat file could be used to trace the infection path of a virus as well as any other characteristics of the virus. Anti-goat technology provides protection by detecting that it is being executed against a goat file for analysis and changes its mode of execution to either a mode of non-malicious code or attempts to attack the user's system by corrupting files. A virus called "Nexiv_Der", was one of the first viruses to implement the anti-goat protection mechanism (Ször, 1996). The virus was fairly complex, exhibiting both polymorphic and multi-partite (the ability to spread through multiple infection vectors) traits, but was not particularly successful in spreading due to a number of internal bugs.

2.4.6 Code Armouring Summary

While Code Armouring is usually effective in slowing down an analyst from gaining a deeper understanding of the malware, as soon as the analyst does bypass the defences and a signature is generated, the malware will forever be detected by antivirus applications (Brand, 2010, Sikorski and Honig, 2012). It can, as a result, be said that Code Armouring does not provide an active means of evading antivirus applications, instead it provides a passive means of defence which can provide it with more time to accomplish the task for which it was created. The previous subsections also demonstrate that malware authors are very aware of what methods analysts are employing against their malware and in response, they are building countermeasures to make the analysts' jobs harder. A number of the techniques described above i.e. anti-disassembly and anti-debugging were not built for malware alone, but were also used by commercial software development companies to protect intellectual property, though as with malware authors, all this can do is slow down the person trying to gain access to the intellectual property. The section to follow will introduce the evolution of active defences as employed by malware.

2.5 Early Malware Defence

Early virus writers did not attempt to evade antivirus applications (Rad *et al.*, 2012) because the antivirus industry did not yes exist for the consumer market. Instead,

specific applications were crafted to either inoculate or remove a virus on a case by case basis. Since malware did not have a specific antivirus application to evade, most malware authors attempted to hide their actions from system administrators and users that used the systems which they attacked.

2.5.1 An Early Example

One of the first viruses that attempted to avoid detection by a user was the Brain Virus (Hypponen, 2011, Parikka, 2007) which was first detected in 1986 (Hypponen, 2011). The virus worked by first moving the boot sector to a different part of the disk and then overwriting the boot sector with the virus. Instead of getting an error message when trying to load the disk (because of a corrupt boot sector) the virus would intercept the call and redirect any access to the new boot sector location (OECD, 2009). With this in place, most users would not know they were infected until the virus showed its alert message. This alert message was intended to let the user know they needed to contact the authors for support. While this may seem odd, the Brain Virus was not originally written as a virus but as an anti-piracy protection scheme that would track users that used their software without purchasing it (Parikka, 2007).

2.5.2 Response

There was no direct response to the Brain Virus from the antivirus industry, as the number of viruses in the wild during the late 1980's did not require a generic method of removal. Solomon (1993) provides a list of only 8 - 24 viruses that were circulating at the time. To put this in perspective, this occurred approximately three years after the Brain Virus was first encountered. The listing also demonstrates that the common method for disinfection was to simply provide a simple technical walk through to the system administrator who could then remove the offending application from their system. Although this may have been the case between 1986 and 1987, by 1988 antivirus companies started forming and released a number of products aimed at the removal of a number of known viruses.

2.5.3 Evolution

Even though the Brain Virus and others like it did not attempt to evade antivirus engines they did begin the chain of malware evasion techniques (White *et al.*, 1995, Parikka, 2007). The next technique in the chain came in the form of malware encryption. This technique (which will be discussed in the next section) is the first documented technique that actively tried to evade antivirus engines instead of just hiding itself from the user on the machine.

2.6 Encryptors

Before detailing the history and evolution of encryptors, there are a number of points that need to be brought forward for this section as well as the work in the sections to follow. Regarding the encryption referred to in this section, as well as the remaining sections, the encryption being referred to is the encryption of the internals of a binary both at rest and runtime (Nachenberg, 1997). It is not referring to the encryption of user data, which is a malicious side effect of a number of malware applications (Young and Yung, 1996).

2.6.1 History

Following the Brain virus, encryption was the first major evasion technique that was used. Encryption is the process of converting text into a form known as ciphertext which cannot be understood except by the intended target of the message. The strength of an encryption technique is largely dependent on the following combination: the size of the key and the implementation being used as described by Filiol (2004). When dealing with malware that has been coded to use encryption, malware authors started with a simple XOR cipher encryption as described in the work by Young and Yung (1996). By using a simple technique, the margin for error occurring when developing an encryption algorithm was reduced. Large keys by comparison are easier to build and this in turn, increased the difficulty of decryption through brute force. An example of this in current malware is the Gauss malware (Goodin, 2014a,b). The Gauss malware currently has an encrypted payload which malware analysts have not yet been able to decode. While the analysts working on the malware have worked out the encryption scheme utilized, they have not been able to force the key or even guess the key, simply because the key is so large and unique . Even though the encrypted payload is not being used to evade antivirus engines, it does prevent malware analysts from determining the final target of the malware.

2.6.2 An Early Example

This technique was first noticed in the wild around the year 1986, in the event of the Cascade virus (Hypponen, 2014). The virus operated by encrypting the body of the virus with an encryption scheme of the author's choice. The Cascade virus implemented a very simple means of encryption called an XOR cipher. The XOR cipher, also known as an additive cipher, works by adding a cipher key to the plain text to produce ciphertext (Tutte, 2000). To extract the plain text from the ciphertext, the key is subtracted from the text, which results in the output of plain text again.

The XOR cipher does present a problem. If the key is small enough, a cryptanalyst could use statistical analysis of the text to determine the plain text. To prevent this, the

Cascade virus attempted to make the encryption stronger by using a dynamic key for the encryption process. This dynamic key that was used is based on the length of the body of code. This meant that the key used by the virus would change for each binary file that it infected. Additionally, the Cascade virus was also one of the first of a few viruses that implemented Code Armouring. The armouring of the code meant that the job of the malware analyst tasked with analyzing the virus was made more challenging, which in turn meant that the virus had an extended lifetime. The Cascade virus implemented armouring by using the stack pointer which would change as the application ran as part of its cipher key.

2.6.3 Response

Antivirus vendors were able to adapt fairly easily to the rise of encryption in malware. They were able to counter the increase of encryptors by detecting the decryption routines used by the malware. The early method of detecting decryption routines (that will be covered in further detail later in section 2.11.2) is through the use of entropy detection which is introduced as the second generation of antivirus detection methods (Davis, 2009). This was not used widely by the antivirus engines of the time, but it has become more prevalent in antivirus engines today since the current processing power available to modern antivirus engines is significantly higher than the past. The entropy calculation can execute substantially faster, making this method of detection notably more feasible. The negative result of entropy detection is that it is prone to false positives if a binary is responsible for a significant amount of dynamic changes to itself.

2.6.4 Evolution

Encryption is the first method that was built specifically to allow a malware to evade an antivirus engine. The technique itself had a sort of internal evolution as malware authors attempted to make their encryption routines even harder to detect and implemented more complex encryption routines. Unfortunately, because encrypted malware always requires decryption before it can execute its code, this decryption routine becomes a weakness as it is a single point of failure (detection). Oligomophism which is the next technique in the chain of evasion techniques will attempt to fix this problem and will be explained in the next section.

2.7 Oligomorphism

Oligomorphism, which is part of the encryptor family of evasion techniques, was a direct result of antivirus applications being able to detect the decryption stubs used by viruses at the time (Ször, 2005). These decryption stubs would still allow the viruses to employ encryption to hide the body of the virus but would generate different decryption stubs each time it spread. These decryption stubs would employ different looping techniques that, in turn, generated different byte code that looked different each time a new copy was created. This meant that antivirus engines could not detect the decryptor stub with the use of static signatures (Schiffman, 2010, Rad *et al.*, 2012).

2.7.1 An Early Example

The first time oligomorphism was detected was in the Whale virus in the 1990s (Skulason, 1990b, Ször, 2005, McAfee, 2014a). While the virus was an evolution of the usual encryption/decryption technique, it still suffered from a flaw in its design. The fatal flaw was simply the limited number of decryptors used by viruses that employed oligomorphism (Ször, 2005, Schiffman, 2010). This meant that with a sufficient number of runs, a malware analyst could generate signatures for all the decryptors a virus author had implemented. While this may seem like a simple enough solution and method for detection, viruses like the Memorial virus had 96 different decryption stubs built into it, which meant 96 generated signatures needed to be generated as noted by Konstantinou and Wolthusen (2008).

2.7.2 Response

The antivirus vendor industry did not need a specific method to manage the detection of oligomophism evasion techniques as the signatures generated by the decryption stubs were still static for most of the viruses. In the instances where there were multiple decryption stubs, the antivirus companies iterated as many decryption stubs as possible, and made a record of their signatures. With multiple signatures linked to a single piece of malware, an antivirus engine could still, relatively reliably, detect any malware that employed oligomophism to hide itself. In certain cases (i.e. Memorial virus) generating multiple signatures is not a requirement when the signature created is a nearly exact match. The same applies to creating a wild card signature, as both methods are simply means of creating generic signatures for malware based on multiple points in a file. A more detailed discussion on how these detection methods operate will follow in section 2.11.1.

2.7.3 Evolution

The use of oligomophism was a step forward in terms of the evasion techniques employed by malware authors and although the technique served its purpose as an intermediate solution, it was slightly flawed. Its flaw lay in the limited number of possible stubs it could generate without bloating the malware. Additionally, the decryption stubs that were generated lacked a dynamic nature. In response to this problem polymorphic viruses were built as described in the next section.

2.8 Polymorphism

Polymorphism builds upon and attempts to fix the challenges presented in the previous section on Oligomorphism. Nachenberg (1996), Konstantinou and Wolthusen (2008) describe polymorphic viruses as viruses that were able to drastically modify themselves between each iteration. Later, polymorphic viruses that were detected included mutation engines. These engines, when implemented correctly, allowed the virus to generate millions of slightly different iterations of itself (Nachenberg, 1997).

2.8.1 Early Example

The first known record of a polymorphic virus was the V2PX virus which was written as a research project by Mark Washburn (Rad *et al.*, 2012, McAfee, 2014b). The purpose of his work was to demonstrate to Antivirus vendors that they could no longer depend on static signatures to detect viruses. The V2PX worked by inserting random calls to functions that would not affect its code but would alter how the virus looked to an Antivirus engine (Lammer, 1990). An example of a function that would not affect the execution of the code is the NOP instruction. The NOP is a "nooperation" instruction, it is so named because on encountering a NOP instruction, the CPU does nothing and moves on to the next instruction. This instruction provides a malware author with a versatile and simple way to change the signature of a virus without affecting its internal working operations (Akritidis *et al.*, 2005).

The key to effective polymorphic viruses, as demonstrated by the V2PX virus, is to sufficiently change the program so that it retains little to no similarity between iterations. This was primarily achieved through the use of junk data which performed no function (as mentioned previously) or it expanded single instructions into multi-line instructions (Skulason, 1990a, Konstantinou and Wolthusen, 2008).

2.8.2 Response

The skeleton detection method of detecting viruses which was created by Kaspersky. The skeleton method will be elaborated upon in section 2.11.2. This method stripped out supposedly junk data before scanning a virus. This would allow the antivirus engine to scan the part of the virus that was actually executed.

2.8.3 Evolution

It should be noted that until that point in time, polymorphism had simply been used to make the detection of the decryption stubs harder. This meant that for the most part, the body of the virus that was encrypted would still remain the same. As soon as the main body of the virus was decrypted, it was possible for an antivirus engine to detect the virus through its decrypted signature. The only methods of altering themselves employed by malware authors revolved around the insertion of junk instructions; fake loops; and fake jump instruction calls. Each of these attempts caused the decryption stubs to either grow or shrink depending on the current iteration that the malware had spawned. To overcome this challenge, the malware authors needed to create a piece of malware that was wholly dynamic in that that would change significantly in how it executed between each time it infected a binary. This resulted in the release of metamorphic malware which will be covered in the section to follow.

2.9 Metamorphism

As is the case with each new evasion technique, metamorphism is an evolution of the idea of polymorphism. While polymorphism attempted to make the decryption stub difficult to detect, metamorphism attempted to make the virus itself a harder task to detect without the need to encrypt the virus itself (Konstantinou and Wolthusen, 2008). Metamorphism has been described as a polymorphic body since the body of the virus changes from generation to generation (You and Yim, 2010).

2.9.1 An Early Example

The first known attempt at using metamorphism as an antivirus evasion tactic came in the form of the Regswap virus in 1998 by Vecna (Ször, 2005). The virus implemented metamorphism by dynamically and randomly changing the registers it used within its body between each generation. This method of implementing metamorphism with dynamic registers was only one of the ways in which virus authors implemented metamorphism. Other methods included:

- **Garbage Code Insertion:** The method entailed inserting garbage code by a metamorphic engine which would not change the functionality of a programme but defeat pattern based scanning. This is the old technique used by polymorphic viruses, while it was generally avoided there are still some early metamorphic viruses that used it.

- **Subroutine Reordering:** The Win32/Ghost was one of the many viruses that implemented this method for dynamic code generation (Ször and Ferrie, 2001). This method of metamorphism differs from the garbage code insertion in that the virus code between each version is the same (Ször and Ferrie, 2001). To make the virus seem different between infections, it changed the order of the code in the binary which resulted in a different signature for the virus. Branch tracing is used as the method of detecting this type of virus. Since the code base is always the same, the binary would always have the same branching paths which could be used as a signature to identify the virus (Radhakrishnan, 2010).

- **Instruction Replacement:** This practice was implemented by a number of the mutation engines. This usually involved changing long form calls of an instruction into many smaller calls. The Win95/Bistro virus is one of the many viruses which used this (Ször, 2000).

- **Code Integration:** This process was only really implemented by the Zmist virus. Ször and Ferrie (2001) demonstrate that the Mistfall engine, which powered the virus, was able to decompile a binary and insert its required code into the binary. Once it had added the code it needed, the virus would rebuild the binary, so that it functioned normally. This was not seen in any other metamorphic virus of the time, and speaks to the advanced nature of the virus.

Aside from the Regswap virus that was discovered in the early stages of metamorphic virus evolution, there are two other viruses that have to be mentioned due to the sheer complexity of their code and advanced techniques implemented.

The Zmist (Z0mbie.Mistfall) virus, which was first introduced under the topic of Code integration, was the first of the viruses that broke new ground in terms of the advanced functionality implemented by a virus. The code integration which is the most significant and advanced feature of this virus, set the virus apart from all other viruses at the time by allowing it to integrate with a host binary without the need to change the original entry point of the host binary (Ferrie and Ször, 2001). The virus also implemented Code Armouring techniques by hiding the running process and spawning a new copy of the host application so that it would remain undetected. The virus had no other aim other than

to spread as much as possible and it did so by simply searching for exe files and if, they matched the infection parameters set by the virus, they would be infected.

The second virus that showed significant innovation in the metamorphic space was the Win32/Simile virus (Ször and Ferrie, 2002). In a similar method to Zmist, Simile did not change the original entry point of the application it infected and it also implemented significant Code Armouring techniques which prevented researchers from gaining a full understanding of the virus. To further complicate the analysis, the virus is estimated to have 14000 lines of code, of which a major portion was dedicated to its internal mutation engine. Notable work has been done by Ször and Ferrie (2002), Konstantinou and Wolthusen (2008) on the Simile virus, describing how it implements its mutation engine as well as how the virus manifests itself to the user.

2.9.2 The Rise of Virus Toolkits

While metamorphism remains an effective method for evading antivirus engines, the biggest challenge facing its implementation was that it was relatively difficult to build. As a result, virus authors who had advanced virus building skills would construct polymorphic toolkits that would allow others to simply plug-in their virus code and have it circumvent most antivirus engines with a higher level of accuracy (Pearce, 2003). One of the first toolkits that started this practice of selling antivirus evasion software, was a toolkit called the Mutation Engine (Pearce, 2003). After the release of the Mutation Engine, a few more toolkits were released, the details of some of these engines are listed below:

- **NuKE Encryption Device (NED)** : NED[4] was created by the author of the Virus Creation Lab and released in October 1992 (Beroset, 1993, Fuhs, 1995).
- **TridenT Polymorphic Engine (TPE)**: The first version of TPE[5] was released in December 1992, with four subsequent versions being released to fix a number of bugs. Around the time of it release, viruses created with this specific engine were almost undetectable (Fuhs, 1995).
- **Dark Angel's Multiple Encryptor (DAME)**: DAME[6] was released in June 1993 as a response the NED engine which was released by the competing group known as NuKE. The engine was published with fully commented source code instead of a compiled format (Angel, 1993, Fuhs, 1995).

[4] `http://vxheaven.org/vx.php?id=en02`
[5] `http://vxheaven.org/vx.php?id=et06`
[6] `http://vxheaven.org/lib/static/vdat/engine1.htm#DAME`

2.9.3 Response

The antivirus vendors had been slowly building up their defences against malicious software, but were only able to detect malware that employed the metamorphic evasion techniques based on the payload the malware carried (Ször and Ferrie, 2002). The payloads themselves were again only detected in cases where the payload was destructive in nature. If the payload implemented an unknown exploit, it is very likely that it would evade all antivirus engines entirely. In fact, authors preferred to get recognized for their elegant work instead of being acknowledged for the malicious consequences that could result - as noted by the author of the Simile virus (Petik, 2002).

2.9.4 Evolution

The metamorphic evasion technique itself did not have any further evolution in its own right. This is because for the most part, analyses of various metamorphic techniques have shown that, when implemented correctly, the malware will evade antivirus engines entirely. The challenge lies with implementing a metaphoric shell in order to wrap a generic piece of malware. It is extremely complex to write a metamorphic virus even with the internal knowledge of how the virus should work. Extending this to a generic piece of malware, where there is no foreknowledge, is entirely error prone, as the malware may already have Code Armouring techniques in place to prevent modification, or be encrypted in which case any modification would break the decryption process.

2.10 Packers

Packers were not originally built for use in malware, instead they were meant to compress applications so that they could be transferred faster over the Internet (Harper *et al.*, 2011). The use of packers increased once malware authors discovered that packers could easily allow their existing code to evade antivirus engines (Perdisci *et al.*, 2008). It is important to note that this would usually only work against antivirus engines that scanned for viruses on access (O'Kane *et al.*, 2011). Heuristic detection (which would come along at a later time) and would allow antivirus engines to detect the viruses after being unpacked.

A packer is normally built in two parts, this is similar to the way encryptors operate. The author of the malware creates a small packer binary which will pack a target binary of their choice. Once the target binary is packed, the decryption stub is injected into the final packed binary. This is similar to the way encryptors originally worked (Brulez, 2009). The entry point for the binary is then changed, so that the decryption stub is run first so that the application can be extracted into its original form and then run as

normal.

Packers still suffer from the same issues as the early encryptors did: the unpacking stubs are easy to detect (O'Kane *et al.*, 2011). Once an antivirus application detects a packer stub, it usually flags an application as malicious regardless of the packed content. In most cases this means that many legitimate applications are falsely flagged. In order to prevent false flagging, certain antivirus vendors will take the extra step and run the application through an emulator when a packer stub is detected. This will allow the application to run and allow the antivirus to detect malicious behavior.

Oberheide *et al.* (2009) point out that packing is often more of a time hindrance to analysts instead of a deterrent. It requires a time consuming manual analysis to be performed against them before details about the malware can be extracted. The similarities of packers employing similar techniques with slight modifications to them as the authors find issues with the original technique is described by Guo *et al.* (2008) in which they discuss the problem facing the security industry having to deal with packers. The extent to which packers are effective was demonstrated by Oberheide *et al.* (2009) with his service PolyPack (Oberheide *et al.*, 2009): an increase of 4.83 times better evasion was recorded when compared to the baseline binaries that were presented.

2.10.1 An Early Example

A brief listing of the early packers and when they were released can be seen in Table 2.1 (Brulez, 2009). Two of the packers seen in Table 2.1 UPX and PECompact, are still available and actively maintained. Based on the estimated release dates, it can be concluded that UPX and PECompact have been in circulation for approximately fifteen years. This will be noted when they are used for testing the packing evasion techniques in Chapter 5. Unfortunately, aside from the release dates for early packers there is little recorded history of how the packers and encrypters in table 2.1 worked.

Table 2.1: Packer Early Release Listing

Packer Name	Version	Release Date
Stone PE Crypter	1.0	23 December 1997
PECRYPT32	1.01	22 January 1998
PELOCKnt	2.01	7 April 1998
Petite	1.0	22 May 1998
UPX	0.50	3 January 1999
PECompact	v0.91 beta	31 July 1999

2.10.2 Response

Guo *et al.* (2008) notes that, as packers evolved, antivirus engines started detecting the packer signatures (where present) as malicious. As a consequence, many packed executables appeared as malware. While there have been a number of novel approaches proposed to rectify this problem of simply flagging the stub as malicious (see Guo *et al.* (2008), Royal *et al.* (2006), Kang *et al.* (2007)), it has proven difficult to verify which approaches are in use by antivirus companies since the process that triggers these methods rely on runtime detection.

2.10.3 Evolution

Packers in themselves have not undergone any vast changes from what was originally implemented. Most of the efforts have been focused around Code Armouring instead of antivirus evasion. Roundy and Miller (2013) have provided a detailed explanation of the methods implemented by packers.

2.11 Malware Detection Mechanisms

This section aims to present some of the methods employed by antivirus scanners to detect malware. These mechanisms will be discussed in direct relation to their related evasion techniques. According to Ször (2005), the various detection mechanisms can primarily be classified into two generations of scanners, namely first and second generation scanners.

2.11.1 First Generation Scanners

The first generation of scanners were developed and mainly used during the evolution of virus writing during the early 1990s (Ször, 2005, Bustamante *et al.*, 2007). These methods of detection were focused around different means of signature detection. Consequently, while they were useful at the time when they were invented, they quickly fell away to more efficient methods of detection. The list below is a reduced list of the known antivirus methods of scanning and indicates the order in which they were introduced to antivirus applications. The sequence is based on the work by Ször (2005).

- **String Scanning:** String scanning is one of the oldest known methods of detection (Kephart and Arnold, 1994, Ször, 2005). It works by having an antivirus company record a known unique string that would occur in a virus and then it distributes it as part of their signature database (Thengade *et al.*, 2014). The antivirus application on the end users computer system subsequently scans all applications for this string

pattern which would, in turn, indicate the presence of a malicious application. String scanning is still used in modern antivirus applications but to a lesser extent as there are better alternative methods available. This method of scanning is limited by the fact that it can not detect new viruses which have not had their signature recorded (Thengade *et al.*, 2014). The string scanning method of detection also means that dynamic viruses that are able to change their signature can bypass this means of detection fairly easily (Thengade *et al.*, 2014).

- **Wildcards:** The Wildcard scanning method is an improvement of the basic string scanner (Ször, 2005, Thengade *et al.*, 2014). Where the string scanner looks for a fixed string to match, the Wildcard scanner looks for string matches but allows for variations in the signature. This primarily allows the antivirus engine to deal with multiple variants of a virus creation group (Thengade *et al.*, 2014). The types of Wildcards vary from simple byte replacements to byte ranges and in the case of some scanners, regular expressions. This method of scanning suffers from the same limitations as the Simple String Scanning in that it is limited to detecting viruses that matches the search pattern (Thengade *et al.*, 2014).

- **Bookmarks (Check Bytes):** The Bookmark method of detection is more of a modification to the two previous detection methods presented above. The changes to the simple scanning and wildcard are implemented to make them more accurate which in turn ensures that there are fewer false detections (Ször, 2005). The Bookmark method of detection records the location of the string signature as an offset from the start of the virus. In this manner, when the antivirus engine needs to look for a signature, it can jump to a specific location. Alternatively, the antivirus application can compare the length from the start of the virus body to the signature that was found in order to determine if a virus or a false positive was found (a false positive would have a different bookmark length). By looking at specific locations for a signature the antivirus application does not need to scan the entire application, this in turn speeds up the scanning process (Rad *et al.*, 2011).

- **Top and Tail Scanning:** Top and Tail scanning refers to the locations that an antivirus engine would scan (Rad *et al.*, 2011). Top refers to the header of a file and is usually considered (at most) to be the first eight kilobytes of a file. The tail conversely refers to the ending of a file, and is also generally limited to eight kilobytes (Ször, 2005). This type of scanning was implemented to increase the speed at which the antivirus could perform since (as computers grew more powerful) the biggest increase came from large file sizes as well as slow disks. By searching sixteen kilobytes of data only versus two megabytes of data, an antivirus engine could cover

many more files in the same time frame it would take to scan a single file. It should be noted that this method of scanning still requires the application to look for a fixed string or Wildcard signatures in the blocks of the application it scanned (Ször, 2005, Rad *et al.*, 2011).

- **Fixed Point and Entry Point Scanning:** Fixed point and entry point scanning represents the evolution of the techniques listed above. These two methods still work on signature detection, but instead of scanning chunks of data and hoping to find a signature, the fixed point scanning simply looks at exact points in an application for a signature (Ször, 2005, Rad *et al.*, 2011). Entry point scanning took this idea a bit further and parsed the binaries header and looked for the entry point location (the entry point is the location of the code which bootstraps and launches an application). Once this location was found it was scanned for known virus signatures. It should be noted that these techniques worked because viruses would often replace the entry point for an application to redirect the control of the application (Xu *et al.*, 2007).

- **Generic Detection:** Generic detections were mainly used for variants of a known virus and generally consisted of multiple previous techniques with some code to tie the techniques together (Ször, 2005). For example: an antivirus company might use entry point detection to get to the initial area where the virus is located and then search for Wildcard data in the area around this entry point.

2.11.2 Second Generation Antivirus Scanners

The second generation of antivirus scanners is a general grouping applied to the detection mechanisms that was implemented to deal with the rise of dynamic evasion techniques (Chien and Ször, 2002, Bustamante *et al.*, 2007). The following list outlines the major techniques which the second generation of scanners used.

- **Smart Scanning:** Smart scanning was the first of the techniques to be developed as a response to malware authors distributing malware kits which would allow a prospective malicious user to generate a custom virus based on the variables setup and available to the malware kit (Ször, 2005, Leder *et al.*, 2009). The way in which most malware generation kits worked was by changing the order of function calls or generating random names and inserting junk data into the calls. In this manner, every application would look different but at its core was the same. Smart scanning worked by ignoring all the junk data and looking for signatures that matched bits of code that was fundamental to the application. This technique was also significantly

used to scan scripts that would have a malicious outcome. Even if the script added spaces and new lines, the smart scanner would ignore these and look for the code sections and then scan code.

- **Skeleton Scanning:** Skeleton scanning is a method of scanning invented by Eugene Kaspersky (Ször, 2005)and works in a similar method to smart scanning. It discards non-essential code and whitespace. Instead of then looking for a signature in the remaining code, the scanning application built a signature of how the code was structured. This code structure was then compared to previous known structures. This process of detection further enhanced the ability of the scanner to detect any variants of viruses. By defining what was the fundamental to the virus in a signature, the antivirus company could easily ignore any new code added to a variant.

- **Nearly Exact Identification:** Nearly exact identification is similar to generic detection in that a malware analysts would define multiple signatures for a virus that would allow for partial matches (Ször, 2005). This method of scanning was required, in the case of overwriting a virus, for antivirus applications to determine if the application could successfully disinfect an application. Instead of relying on a single Wildcard scan, the signatures would be comprised of a number of Wildcards and static strings. An example of how this would be used is when a variant of a virus is found. The signatures might match but because of an invalid checksum, the application will not attempt to repair the software as it could cause further problematic issues.

- **Heuristic Analysis:** Heuristic analysis relies on being able to monitor an application and then make a decision if its malicious or not (Ször, 2005, Gryaznov, 1999). Unfortunately, Heuristic analysis is not reliably accurate and subsequently results in a large number of false positives which does not benefit the end user (Gryaznov, 1999).

- **Entropy Detection:** Entropy detection is relies on calculating the entropy of an application and making a decision based on the level of entropy found in the code. This method is often used when checking for packers and encryptors since these types of evasion techniques result in applications with a large amount of random data that only becomes valid once the application is executed (Lyda and Hamrock, 2007). This large amount of random data results in a high entropy scores which is why entropy detection was so useful against encryptors and packers (Shafiq *et al.*, 2009, Alme and Eardly, 2010, Saleh *et al.*, 2014).

- **Filtering:** Filtering cannot be construed as a method of detection as a standalone

model, rather it is accepted as a modification to the way in which applications would scan for a virus (Ször, 2005). It worked by limiting the number of files that needed to be scanned based on the type of application or data that could be infected by a virus. As a result, if a virus affected the boot sector the antivirus application would not need to scan all the users' files to remove the virus and further limit where the virus originated based on the type of applications the virus affected.

- **X-Ray Scanning:** X-Ray scanning is another modification of the usual methods of scanning. X-Ray scanning attacks the encryption and packing systems implemented by malware authors and then checks the decrypted or unpacked data for a known signature (Alme and Eardly, 2010). It is known as the X-Ray technique because it allows the antivirus application to look into the software for signatures that might not otherwise be easily detectable.

- **Code Emulation:** Code Emulation is the last of the second generation scanning techniques and is also possibly the most complex technique. The antivirus application would emulate a number of instructions that the Central Processing Unit implemented, and then start tracking these instructions (Alme and Eardly, 2010). By doing this, the antivirus application would be able to determine if the application is malicious or not based on the calls it was making as well as the areas of memory with which it was interacting. This method of detection is very closely related to Heuristic analysis, but looks at the application on a much lower level.

2.12 Related Work

The taxonomy of techniques that are used to evade antivirus engines primarily stem from the seminal work by Ször (2005). He goes into detail explaining how each of the techniques originated and elaborates on their level of effectiveness. Previous work completed in the area of testing antivirus engines showed that there was a lack of research in this area (Christodorescu and Jha, 2004). Their research was initiated after the conclusions of Gordon and Ford (1996) indicated that there is a lack of testing processes and means of safe testing of viruses. Their research also demonstrates methods that are slightly different to ours, in that the services used in this research were not available. As a result, the antivirus scanning was performed by manually submitting the malware to the antivirus engines and recording the results after wrapping them with a selection of obfuscation techniques detailed in their paper.

Oberheide *et al.* (2007, 2008) were some of the earliest authors to explore the concept of automating the process of antivirus scanning and decoupling it in such a manner that

it can scale as needed. Similarly, Oberheide *et al.* (2009) demonstrated in their work on PolyPack how to build a service which could automate the process of evading an antivirus engine through the use of packing. Their research was built in response to Kang *et al.* (2007) who demonstrated that it is possible to automate the process of unpacking malware before being scanned. They also illustrated that unpacked code showed a significant increase in detection by antivirus engines versus packed code. Further work was completed by members of the BitDefender team regarding investigations they performed in respect to moving antivirus engines into the cloud (Chiriac, 2009). The outcome of their work showed that while it is efficient to perform parts of the scan in the cloud, issues around privacy and bandwidth considerations do not make it entirely practical.

In an empirical study of six of the popular antivirus engines at the time, it was determined that the antivirus engines were not able to effectively detect all forms of malware (Sukwong *et al.*, 2011). The study followed the path of previous studies such as those performed by Royal *et al.* (2006), Kang *et al.* (2007), Oberheide *et al.* (2007). This is a deviation from the method of testing used in this report when compared to the work done by (Oberheide *et al.*, 2009, Kang *et al.*, 2007, Oberheide *et al.*, 2007), in that a large number of malicious binaries are not tested in a scatter shot approach. Rather, a carefully selected number of binaries were selected for testing. The reasoning behind this was to determine if a targeted binary could be modified in such a way that it would bypass an antivirus engine, this is known as the observed detection technique (Borello *et al.*, 2010). The online method of submission and recording as used in chapter 4 was initially proposed and then demonstrated by Bishop *et al.* (2011). This method of testing involves submitting the binaries that are being tested to an online service and then recording the results for later analysis. This approach differs from Sukwong *et al.* (2011)'s approach in that an offline lab does not need to be constructed to test each of the antivirus engines. The testing and recording approach was also demonstrated by Haffejee and Irwin (2014) in which the effectiveness of antivirus engines as security layer was evaluated. Further testing using the online method of testing was performed by Swart (2012) in which two antivirus evasion frameworks were tested. The results from the testing demonstrated that a single antivirus could not catch all the evasion techniques, but it was possible for a single custom evasion technique to evade the majority of antivirus engines. Brand (2010) detailed how a number of code armoring techniques protected the binaries they were attached to. This worked also detailed how to detect the armoring techniques and where possible remove them.

Regardless of the approaches used (online and offline scanning) when testing a system, the testing of antivirus engines are generally classified by two approaches, known as black-box and white-box testing (Adrion *et al.*, 1982). White-box testing is considered

when the internal operation of the system are available to the tester because metrics can be extracted on how the system processes the input. Black-box testing, by contrast, is considered when the internal operation of the system is not known to the tester. It focuses on testing the system by comparing the inputs to the outputs of the system. When considering the scatter shot method of submitting hundreds and thousands of binaries in order to a test a system, it can be seen as a useful white-box testing approach: see, for example, Royal *et al.* (2006) with their automated unpacking system. By submitting as many malicious binaries through the system they could, they were able to gain a deeper understanding about the input data as well as how their system reacted to this input. This is in contrast to the approach used for testing later known as the black-box testing approach. In black-box testing, the internal workings of the antivirus applications are not known and all that is of consequence is the output generated by the antivirus applications. In comparison to the work Oberheide *et al.* (2008), the research performed here is not attempting to quantify the amount of malware that an antivirus engine detects as malicious. Rather, it aims to determine whether the binaries which the antivirus already detected as malicious can be made to appear safe.

2.13 Summary

This chapter covered the evasion techniques implemented by malware authors in their malware over time as well as the detection mechanisms the antivirus industry implemented in order to detect and remove offending malware. What has transpired is that the antivirus engines are always a step behind, and simply playing catch up by implementing new techniques for detecting malware once a new evasion technique is found through manual analysis. Furthermore, the malware and antivirus evolution has constantly improved the state of art for both sides as they attempt to outsmart and outdo each other. This chapter also demonstrated why it is important that the antivirus industry has its products constantly evaluated and touches on how it was previously tested. Chapter 3 will explain how this research intends to expand the area of testing by demonstrating the constraints with the existing models of testing. In conclusion, suggestions for updated methods of testing antivirus products will be made in the next chapter.

Chapter 3

Antivirus Testbed

3.1 Introduction

As highlighted in the previous chapter, the antivirus industry needs to continuously evaluate and re-evaluate its products to ensure that it is on par with what is available globally in terms of malware and malware evasion practices. This work also needs to performed by, firstly a suitably technically advanced user and, secondly in a consistent repeatable manner. This chapter aims to disclose the method of testing antivirus engines on a large scale and in a consistent and repeatable manner, as suggested. The chapter to follow will explain how the antivirus applications scanning methods are tested. The goal of this chapter was to determine the most suitable methodology for performing the test in the upcoming chapters. The method of testing needs to lend itself to automation, be repeatable and perform for multiple applications simultaneously.

3.2 System Selection

The options that were considered were grouped into either online or offline services. There are currently no publicly available offline systems that can be used to test antivirus engines in an automated fashion. Previous work analysing metamorphic malware by Borello *et al.* (2010) demonstrates that building offline systems are effective but require resources to maintain. Oberheide *et al.* (2008), provides a good point of departure for the purposes of this study. Regrettably, there is no output from this work that can be used immediately. This means that any custom testbed would need to be implemented from scratch. In contrast, there are a number of systems that are available that can be used for the online testing. Of these systems, Virustotal[1] is the most public and most accessible system and is the one that will be used for testing later in this chapter based on previous work completed

[1] `http://www.virustotal.com`

using the service which has proved that it is effective as an online testing platform (Gashi *et al.*, 2009). The only other comparable service is MetaScan[2], while the website provides the same functionality as VirusTotal it is not nearly as well tested or used by the security community due to it being a very new website (only established on 1 December 2012). All other services that provide online scanning are targeted at specific analysis techniques instead of scanning via an antivirus engine.

3.3 Testing Methodology: The Custom Testbed

Before beginning the tests with Virustotal, a custom testbed will be built to evaluate its effectiveness and viability versus using an online system. The testbed will attempt to perform two major functions. The first of these is to automate the scanning of the malicious software. The second function under scrutiny is the ability for the testbed to scale in size based on the sample count as well as the number of antivirus engines that needs to be tested.

3.3.1 Setup

A virtual environment will be used when building the custom testbed. The requirements for building this environment are listed below.

- **Virtual Machine Environment:** The virtual environment is required so that the system is sufficiently isolated from the tester. This is important in order to prevent infecting the host when testing with malicious binaries.
- **Operating System:** A Windows operating system is required to ensure that the antivirus applications are able to install and execute correctly.
- **Antivirus Application:** A single antivirus application will need to be installed per virtual machine instance. This is to prevent conflicts between different antivirus engines. When testing an antivirus product for effectiveness and viability, two attributes are being primarily tested between the different antivirus products. This is the ease in of automation and the ability to detect malware in an offline mode. The ability to detect malware in an offline mode is crucial as one of the main benefits of offline detection is that the tester can be sure that the results from scanning a file are not submitted to antivirus companies for further analysis, as is often performed by VirusTotal. The ability to automate the antivirus is just as important as the

[2] `https://www.metascan-online.com/en`

need to scan multiple binaries with different evasion techniques applied will take too long if done manually.

- **Browser:** The browser installation is optional per virtual machine. Virtual environments such as VirtualBox, allow a user to transfer files between the host and a guest using internal processes. If this process for transferring files to the guest is not used (or not available in the selected Virtual Machine Environment) then a browser is the next simplest way to transfer files into a virtual environment without the need to install third party services such as a FTP server.

- **Automation Language:** The automation language can be any programming language that is able to control the Windows environment. This is required since many antivirus applications do not expose a programming interface or provide a command line version of the application.

For the base virtual environment, Virtual Box was used for building the virtual machines (VMs) as it is free and supports all the operating systems (OS's) that can be tested. The operating system that was selected was the Windows operating system which was installed in Virtual Box using the default install options. The operating system was then updated to ensure that all available patches were applied. Once this was completed, a snapshot of the operating system was taken using the built in Snapshot feature in Virtual Box. This allowed for rollbacks to a known good state after executing a potentially malicious binary.

At the time of comparison the top three antivirus engines based on protection were Avira, Bitdefender and F-Secure as listed on the AV Comparatives website[3]. Of these three antivirus applications, Bitdefender[4] was the simplest application to automate. Further Bitdefender provides a free version of their application for personal use which was the version that was used for testing. The free copy also provides a non crippled version of the core scanning engine (which analyses the binaries) which is what is important in the upcoming tests. Due to time constraints the remaining two applications were not setup and automated, as such after the base system was setup, the latest version of Bitdefender was downloaded from the official website and installed.

After the antivirus was installed, a full system scan was executed with the Bitdefender to set the system in a known good state before the malicious binaries are submitted. This was also required since Bitdefender requires a full scan before it allows for a selective scan to be performed. The virtual machine also had its network connection set to 'host only' mode ('host only' mode allows connections only to the host virtual machine) during scans

[3] `http://www.av-test.org/en/antivirus/home-windows/windows-8/april-2014/`

[4] `http://www.bitdefender.com/`

to ensure that the local virus engine is performing the scans and not sending files to a remote server for further analysis.

3.3.2 Core Application Installation

Next a number of supporting applications were installed. First a copy Python 2.7 was installed, this helped with scripting a number of tasks later. Lastly a copy of Firefox and Autoit were installed. The Firefox installation was optional, but it assisted in the acquisition of a number of applications when a download was required.

3.3.3 Automating Scans

Owing to recent updates[5], it was not possible to perform a command line scan using Bitdefender. Consequently, the scan are automated internally within the Virtual Machine using Autoit[6]. Autoit is a scripting language that allows a user to script control over the Microsoft Windows graphical user interface. An example of scripting the user interface would be finding a window by handle and then moving the mouse cursor to click on the window that was found. The process for automating the scanning of files is outlined by figure 3.1, this shows the high level process that will be applied to a binary when it needs to be scanned.

Automation Details As indicated in figure 3.1, the file will be uploaded to the virtual machine via a web transfer using a simple web application hosted in python. Once the binary is transfered to the virtual machine the network interface may be disabled if host only support is not provided by the virtual environment being used. A manual scan will then be initiated which will begin by opening Windows Explorer at the location at which the file was uploaded by right clicking the opened folder and selecting the scan option. Once a scan is completed, the results will be collected from Bitdefender's log location. The logs contain the scan results which will be parsed from the XML[7] markup language and stored for later reference in a SQLite[8] database. The Autoit script that will be executed when a user wants to run a scan on the virtual machine, then automates the scanning by launching the "Windows Explorer" application. The script then navigates to the folder where the binary which needs to be scanned is located. At this point, the script right clicks on the folder and then clicks on the "Scan this folder with Bitdefender " button. This initiates a manual scan with Bitdefender. Once the scan is completed a pop-up

[5] http://forum.bitdefender.com/index.php?showtopic=17457
[6] https://www.autoitscript.com/site/autoit/
[7] http://www.w3schools.com/xml/
[8] http://www.sqlite.org/

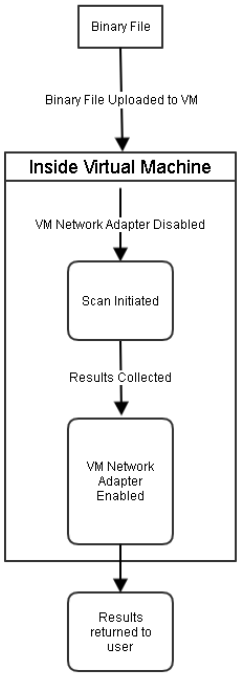

Figure 3.1: Process by which binary will be scanned

notification will appear when the scan is completed. An update to the logs maintained by Bitdefender will also be made. These results are then uploaded to a simple webservice running on the host system as means to get the data back to the host system.

3.3.4 Resource Consideration

The following are some resources that need to be accounted for when planning to setup a custom malware testing laboratory.

- Setup Time
- Execution Time
- CPU Power
- Disk Space
- Financial Costs

While most antivirus engines do provide trial versions of their applications, these usually have time limitations which creates a further burden on the tester who now has to complete the tests in the specified time or to build the testbed in such a way that resetting the testbed back to the original install date is possible. Another consideration is that if

the testbed is being built on a cloud provider to work around the cost of purchasing a valid license key, then resources need to be allocated to pay for the hosting costs. If the cloud hosting resource is not going to be used, then it should be considered that there is a physical cost with acquiring hardware that is powerful enough to test multiple antivirus engines in parallel. If existing hardware that is already in place is used, where speed is not important to the tester, then it should be noted that the scan time will be significantly expedited.

3.3.5 Custom Testbed Summary

In conclusion, while this method of testing the antivirus engines succeeds and reflects more accurately what an end user would be using, unfortunately this process is slow, error prone and resource intensive in comparison to using a dedicated testing service (which will be covered in the next section). In terms of resource usage, there are a number of resource considerations that needed to be accounted for when setting up a laboratory. Resources that need to be factored into setting up a laboratory are listed in section 3.3.4. As a result of the restrictions imposed by these resource requirements, the custom testbed implementation will not be further explored.

3.4 Testing Methodology: VirusTotal

VirusTotal was selected as the online application that was used for testing for a number of reasons (Gashi *et al.*, 2009). The VirusTotal platform is free for public use, this is an important facet as it means that any future researchers will have access to the same tools with which to re-execute the tests. VirusTotal also provides access to an application programming interface (API) by which custom tools can use to build around the platform. This API is what will be used later in this chapter to perform the tests.

3.4.1 Base Tool Compilation

The tools that were built were simply built around the existing Virustotal API and were based on the existing VirusTotal Python library[9] located from the Virustotal Documentation[10]. Based on the code provided by this library, a number of scripts were built to automate. The names of the scripts that were created and their functions are listed below while the code can be found in the appendix.

[9] `https://github.com/Gawen/virustotal`

[10] `https://www.virustotal.com/en/documentation/public-api/`

Table 3.1: Custom Scripts Used For Testing

Script Name	Function
scan.py	Manages the file submission and results recording.
rescanner.py	Initiates the process of rescanning a binary with any updated AV engines, the binary does not need to be available locally.
reportparser.py	Manages the parsing of reports that is generated from the scan.py script for data export.
generatePivot.py	Script used to generate the pivot graphs for the summaries at the end of the packer and encryptor chapter.

3.4.2 Resource Consideration

Fewer resources need to be considered when working with online systems. The resources to consider are Bandwidth Usage, API Limits and File Size Limitations. When using an online system the researcher needs to ensure that they always have a stable Internet connection. Issues including bandwidth and privacy were covered in the work by Chiriac (2009). In conjunction with this, the researcher needs to be aware that, depending on the number of binaries tested, a sufficient amount of bandwidth is available to upload these binaries to the online system. Aside from bandwidth limits, the only other major resource restriction is the API access limits imposed by the online system. In the case of VirusTotal, they limit API requests to four per minute, this means that the scanning manager needs to ensure that this limit is respected or scan requests will fail. Four requests a minute may not seem like an obstacle, but since scans were usually completed in under five seconds there is an implicit waiting period that needs to be considered when scanning more than four binaries. While this is not an issue in most cases, most online systems have upload limits for the file size and as a result very large files cannot be scanned. While it is not a resource consideration, it should be noted that VirusTotal does not implement the full antivirus engine for use when scanning binaries. Instead, only the on demand scanners are implemented and used when scanning. This means that while certain desktop antivirus engines will detect a binary as malicious after a period of execution due to the use of heuristics. Consequently, VirusTotal will not be able to use this method of detection and can account for differences in results between and online and desktop scan.

3.4.3 VirusTotal Summary

In conjunction with the scripts developed to automate the interaction with VirusTotal, the efficiency to cost trade-off means that using VirusTotal allows for the research to be

executed in a speedy and efficient manner. By using VirusTotal, the research can also focus on setting up and testing the evasion techniques instead of trying to determine if the antivirus testbed is working correctly.

3.5 Summary

This chapter dealt with different methodologies by which the antivirus applications could be tested. It also demonstrated that, while it is possible to create a custom offline testbed, this method is not ideal for testing a large number of antivirus engines - unless the tester has significant resources with respect to both time and money. In the end it was determined that implementing the testing via an online service (where the samples are submitted for scanning) was the most efficient and returned results that matched those from the offline test. Having covered how the antivirus applications can be controlled and tested, the next chapter will explain how the antivirus detection engines will be tested in a manner that has both a logical and measurable outcome.

Chapter 4

Antivirus Test Process

4.1 Introduction

Chapter 3 demonstrated how the antivirus testbed would be constructed. This chapter will introduce the processes that will be used to test the antivirus detection engines against each of the evasion techniques. This process involves submitting binaries to an antivirus engine and recording the results. Since the binaries that are submitted can be controlled, there is an expectation of what the output should be. Based on these expectations it can be determined if an antivirus engine is successful in detecting the binary as malicious or if it fails by marking it as safe.

4.2 Goals

The goal of this chapter is to find suitable binaries that can be used to evaluate the antivirus engines for the expected outcomes set out in the introduction. The suitability of a binary is determined based on the test cases in Table 4.1.

Table 4.1: Expected Test Case Outcomes

Test Case	**Expected Outcome**
Benign	Should not be detected as malicious or trigger any warnings.
Potentially Unwanted Program	Should be detected as malicious or trigger warnings.
Malicious	Has to be detected as malicious.

4.3 Selection Process

There are currently three categories that are covered when selecting which binaries with which to test, namely benign, potentially unwanted applications and lastly, malicious

software.

4.3.1 Benign

The first category is the benign application. These are applications of which it is certain that there is no malicious intent from the application. The application selected for testing in this section is signed by Microsoft which substantiates the credibility of the application. In later testing, for each of the evasion sections, a number of tests will require a benign application that is known to be non-malicious. The application selected to perform this task is the "cmd.exe" or command interpreter from Microsoft Windows 7. It was selected because it has no other dependencies and is a compact application. While the calculator application is often used as a test application, there are a number of dependencies which will cause it not to execute when running on a remote system if it is not in the correct folder (the Windows system32 folder where it is originally located). Table 4.2 shows that on submission to VirusTotal for scanning, the application does indeed not get detected as malicious by any antivirus application.

Table 4.2: Benign Baseline Scan

SHA256	17f746d82695...
File name	cmd.exe
Detection ratio	0/55

As expected, when submitting the application to VirusTotal for scanning, the results show a zero detection rate which is expected. When wrapping the binary with each of the evasion techniques it is expected that the results should remain zero when submitted for scans. If the detection results increase, it can be attributed to the antivirus detection of the evasion technique instead of the base binary.

4.3.2 Eicar Tests

The European Institute for Computer Antivirus Research (EICAR), as the name states is an organization that developed a signature that could be used to test if an antivirus engine was working. The signature created can be saved to a file, known as the eicar test file[1]. While the eicar file is useful to determine if an antivirus application is running, it is not a reliable means of testing any kind of binary dependent evasion technique. This is because the eicar file needs to have any padding when scanned by an antivirus application. Furthermore, any binary modification technique (of which will be tested in this research)

[1] `http://www.eicar.org/86-0-Intended-use.html`

depends on a valid Portable Executable binary[2]. Since the eicar file is just a string, it cannot be packed, encrypted or modified in any manner. Subsequently, the eicar file will not be used for any testing for the purpose of this research.

4.3.3 Potentially Unwanted Program

The potentially unwanted program (PUP) provides a middle ground for which to test the antivirus applications. These applications have the potential to be malicious, but by nature they were not designed with malicious intent as the end result. These applications work well as a test case, since tests can determine if the antivirus application is looking for a simple signature or it is performing more complex scans.

Before each of the evasion techniques can be tested and a baseline set, a sample malware application needs to be crafted in order to demonstrate that the evasion techniques being tested are indeed allowing the application to bypass the antivirus. For a PUP to be considered for use in later tests it needs to meet the following criteria:

- It must have the ability to be detected by multiple antivirus vendors as malicious. This is needed to ensure that the application is indeed detected and not simply an anomaly.
- It must be safe for the person running the tests. In the instance of accidental execution it must not provide an attacker with access to the PC of the person running the tests. This is also useful for ensuring the simplicity of the tests as a sandbox environment is not needed.
- It must must have source code available. This is required for certain evasion techniques that work by dynamically changing parts of the application's internal working as part of the application.

Based on the above mentioned criteria, the initial application that passed the selection process by matching all of these properties is the NetCat application. The next section will provide an overview of the NetCat application and the rationale supporting the choice for this research.

4.3.4 NetCat

NetCat is an inert application that is not malicious by design. Its core function is to assist in performing various network related tasks. Owing to its ease of use and its small size, NetCat was commonly used by malicious attackers to either bypass firewalls or setup

[2]The file format required for file execution on a modern Windows operating system

reverse connections for the attacker to a system they controlled. Once NetCat became a common vehicle in malicious attacks, the majority of antivirus applications started marking the binary as a virus. Presently, its detection has largely declined. The scan results in the next section documents the instance where a compiled NetCat binary for Windows was scanned.

4.3.5 Netcat Details

Before compiling a custom version of NetCat on our system, it is worth scanning the compiled binary version of NetCat that is distributed along with the source code. This is the version most antivirus scanners are likely to encounter. The variant of NetCat to be used for the base line testing can be acquired from Jon Cranton's website[3] and the zip file can be confirmed with the SHA1 hash of "2d3026b4630789247abf07aa3986d7a697cf4cd" for the complete zip file. Note that the unsafe binary which has remote execution enabled will be tested, as this is more likely to be detected as malicious by an antivirus engine. The option that makes this version unsafe is the "-e" option. This option is usually what causes the application to be detected as malicious because it permits an arbitrary application to be executed once a connection is made to NetCat.

4.3.6 Scanning the compiled binary

Simply extract the archive acquired in the previous paragraph in order to obtain a copy of the compiled binary . Once extracted, a binary called "nc.exe" can be found in the destination folder. Next: navigate to the VirusTotal website and upload the "nc.exe" application. This can be accomplished by clicking the 'Choose File' button and then selecting nc.exe in the folder that it was extracted to. Once the file is uploaded, the 'Scan It' button can be clicked. By submitting the application to VirusTotal, the following results were returned :

Table 4.3: Scan results for NetCat using precompiled binary

SHA256	7379c5f5989b...
File name	nc.exe
Detection ratio	21 / 46

The results in table **??** are a summary of the information returned by VirusTotal. Located on this summary page is a detailed list of all the antivirus engines that detected NetCat as malicious. The application that was submitted can be confirmed with the

[3] `http://joncraton.org/blog/netcat-for-windows`

SHA256 as provided in the virus total summary. This data can be used to validate that the same compiled binary is being tested. When contemplating the results further, it can be observed that 21 out of the 46 available antivirus engines detected the application as malicious. With a detection rate of greater than zero, the NetCat compiled binary is able to meet the first requirement for use as a sample malware application.

4.3.7 Complications with the compiled binary

Unfortunately, because the compiled binary scanned was compiled by a third party, it is difficult to determine if the binary is indeed safe to run locally. It has to be taken into consideration that the third party who compiled the application may have inserted custom code to change its behavior. To overcome this obstacle, a local development environment is set up to compile the application locally. The application requires both the GCC compiler and Make build application to be installed and requires a custom script to build the application as the default 'Make File' is out-dated. Along with the custom build script, an explanation of how the source code can be compiled on Windows can be found at[4]. Once a local build of NetCat is completed, a re-test of the binary is executed to ensure that it is still detected by multiple antivirus vendors as malicious. When submitting the custom build to VirusTotal, the results in table 4.4 are found.

Table 4.4: Netcat scan results with custom compiled binary

SHA256	087a3c776bde...
File name	nc.exe
Detection ratio	1 / 45

The results shown in Table 4.4, are a summary of the information returned by VirusTotal. Since only a single antivirus vendor detected the application as malicious it will not be considered for further testing. It is also worth noting that the one antivirus vendor that did detect the application as malicious was the McAfee-GW-Edition antivirus. Further McAfee-GW-Edition did not detect the submitted application as NetCat, instead it detected the submitted application as a generic suspicious file ("Heuristic.LooksLike-.Win32.Suspicious.J!89"). Note that unlike with the compiled binary, the hash for this version of NetCat will not match the hash that may be encountered while running this test on a different computer. The reason for this is that the compiler generates a compiled binary that is optimised for the current environment that it is being built on.

[4] `http://www.rodneybeede.com/Compile_Netcat_on_Windows_using_MinGW.html`

4.3.8 Reasons for discontinuation of NetCat

While the initial examination of the NetCat binary showed that it was indeed suitable for use in the tests, execution of each of the tests showed issues that caused NetCat to become unsuitable for use. Unfortunately once the binary failed to trigger multiple antivirus engines as malicious (while using the locally built version of the application) it could not be used any longer as this was the first and most important criteria required for further tests to complete.

The tests show us that in the case of NetCat, antivirus vendors are more likely to flag compiled binaries provided by a third party as malicious, as this is what is most likely to end up on the user's computer. It was also shown that by simply re-compiling an application locally or in a different development environment is enough to beat 99% of antivirus vendors in the case of NetCat. This further points to the fact that in the instance of NetCat, most antivirus vendors are simply relying on a hash signature based on the compiled binary to detect the application and that as soon as this is changed the application is no longer detected as malicious.

4.3.9 New Baseline Selection

With NetCat now no longer usable as a sample for the baseline scanning, a new sample application needs to be selected. In an effort to find an application that can fit the rules that were defined previously, an investigation was done to find current tools that are actively used in the security industry that were often detected as malicious but still provided source code so that modifications could be made to the resulting binary. This resulted in the Metasploit payload generated binary being selected for use.

4.3.10 Metasploit binary

The Metasploit binary that will be used is the output that is generated from the Metasploit payload command. This binary is used to deliver a payload provided by Metasploit to a targeted computer. This is useful for the baseline scanning as it would also allow us to check if an antivirus vendor is checking for a specific payload such as a reverse connection back to a target computer or if they are flagging the binary regardless of the payload. Furthermore, the Metasploit payload generator allows for template binaries to be provided. This can be used to decrease the chance of detection at a later stage when trying to evade the antivirus vendors. It will be interesting to see how antivirus vendors react to the built-in countermeasures provided by the Metasploit payload command. These counter measures are often used in real world testing and may be used to flag

an application as malicious. If this is true, then it will be interesting to see if a malicious application can evade antivirus applications by not using these counter measures.

4.3.11 Metasploit Plain details

While the Metasploit payload application does not provide a compiled application in the same way as NetCat does, it does have a number of templates that it uses to generate the payload binary. Since these are the closest applications to a third party compiled application (they differ because the final version will contain a payload within it) the initial scans will be performed with these templates. Note that the template that will be used for the test is the "*template x86 windows.exe*" binary which can be found in the exe templates folder. As seen in Table 4.5: the template binary is already detected as malicious. Based on the results from the NetCat tests this is most likely because of a known hash and further testing with a locally built version will confirm if this is correct. While not as many antivirus engines detected the application as malicious- when compared to the NetCat application, at least one antivirus vendor did detect the binary as malicious. This means that at this stage the binary can be used for further tests.

Table 4.5: Metasploit Template Original Scan Results

SHA256	640fc87b5754...
File name	ad21c93553af23ecec319c0ea5f11b755acc3342
Detection ratio	22 / 55

4.3.12 Metasploit OPCode details

The Metasploit OPCode tests refer to the raw low level exploit code that is embedded within a binary. Taking this into consideration, after testing the plain Metasploit binary, a copy of the Metasploit binary with custom exploit code embedded needs to be tested as well. This binary with the exploit code is closer to what the antivirus engine would encounter when a malicious attacker tries to deploy malicious code to an end users' system. From the results seen in Table 4.6, an increase in the detection rate is observed. Since this binary is similar to the previous binary (with the only change being the inclusion of the opcodes) it can be said that the increase in detection rates is due to the opcodes.

Table 4.6: Scan Details For Metasploit OP Binary

SHA256	1874c340ba2e...
File name	template_x86_windows_op.exe
Detection ratio	23 / 54

4.3.13 Metasploit custom build details

As previously noted with NetCat, the binary compiled by a third party will not be used for our testing and as a result, compilation from source is needed to ensure that regardless of where the binary is built, it is always detected as malicious. Firstly it is recommended that the build be completed using MinGW[5]. It is noted that although Cygwin does provide similar tools it is more error prone. For the specific instructions on how to install MinGW refer to the section in which NetCat was built. The current version of the tests are being run on a Window 7 machine with MinGW as described earlier. The next step in the process is to download the template file from Github[6] and save it to a file called template.c on disk. There is no 'Make File' option to build the template as the original author may have built the binary with a custom script. As a result, simply running the following command "*gcc template.c -o template gcc.exe*", will build the template binary. Results for template scan can be found in the Table 4.7. While the basic compilation with gcc does provide some favorable results, the number of antivirus engines that detected the binary as malicious is low (2/47). This is most likely due to the fact that the application does nothing at all except crash when it runs. Following that, some machine language instructions will be added which will be used to launch the calculator application. This will allow the application to at least run without crashing and possibly trigger more detection routines. The opcode template compiles to the same binary as would be generated by the msfpayload application when using the template x86 windows.exe binary. Notice that the binaries will not be identical to the byte level as they are compiled on different computers. The template file with the opcodes added can be found in template op.c. The code can be compiled with the following command "*gcc template op.c -o template x86 windows op.exe*". Once the application is compiled, the binary can be submitted to VirusTotal again. This will generate the results which can be found in Table 4.7.

[5] `http://www.mingw.org/`

[6] `http://bit.ly/1uGTASj`

Table 4.7: Metasploit Template Custom Build Scan Results

SHA256	76de5d51b259...
File name	template_gcc.exe
Detection ratio	2 / 47

Table 4.8: Metasploit Custom Built Template Scan With Embedded Opcodes

SHA256	1874c340ba2e...
File name	template_gcc_op.exe
Detection ratio	6 / 47

Reading the results seen in Table 4.8, slightly better detection rates (6/47) are observed. This is better than the initial version of the scan. These slightly lower detection rates work in the favor of future analysis. The future scans are with an evasion wrapper - but a higher detection rate means that the Antivirus engines are detecting the wrapper instead of the base application. The test application cannot be built with the Microsoft compiler as this prevents any potential exploit code from running.

4.3.14 Malicious Binaries

There are a number of reasons why testing with a malicious binary captured from the wild is not ideal. Binaries in the wild will not have their source code available for inspection. This makes it hard to determine the exact evasions that the binary may implement or what malicious code it may be executing. Furthemore, if the malicious binary manages to escape from the test environment, this could affect the host system that the researcher is using. With these downsides, the malicious binaries from the wild still provide a useful test case in that they represent verified malicious binaries and causing them to evade an antivirus application demonstrates the true effectiveness of an evasion technique.

4.3.15 Build Process : Sample Malware and Baseline analysis

This section will explain how to build each of the samples that will be tested against the antivirus applications for the baseline analysis. A detailed explanation will also be provided on how the sample malware was selected.

4.4 Malicious Binary Selection

When selecting which malicious binaries to use for testing, the top 3 unique malware samples were selected from the samples submitted to the "nothink" honeypot service[7]. The service records and provides a malware honeypot as well as the statistics around the data collected. Unfortunately, this is the only service that has made its statistics available to the public, and as a result it is not possible to verify if these statistics are indeed accurate.

Since there are no original names for each of the malware samples, future work will refer to the hashes where possible (as each Antivirus has a different name for the malware). To makes this less prone to error for future work, only the SHA256 hashes are listed here even though the original web page (referenced earlier) used a md5 hashes. While initially it was hoped that the top five malware applications would provide enough variety for the tests, the top five contained only the Conficker malware. Therefore, the three remaining malware that have a very high detection rate were randomly selected. These malware samples were collected via the virus share distribution system[8].

Table 4.9: Malware selection choices

ID	Sha256 Hash	Common Name (Bit Defender)
1	83023e32bb12...	Trojan.Win32.Zapchast
2	5b9508b92a63...	Trojan.Zbot
3	742915067b83...	Win32/Sality.dropper

In the tests that will be executed later, it is assumed that after packing, the binary will simply work as expected. This assumption is fairly naive, but owing to the fact that the tests are performed with live malicious binaries, it is not safe enough to execute the applications after packing as it is not possible to determine exactly what the binaries are doing in the background. While Virtual Machine software will be used while packing the binaries in order to ensure there is at least some level of buffering between the host and the binaries being packed, it is not impossible for the malware to bypass the Virtual Machine's protections and attack the host system. All these issues are really not worth considering when many pieces of malware run entirely in the background and do not display any interface to the user and as a matter of course do not present any user interface features with which to make a comparison.

[7] `http://www.nothink.org/honeypots.php`
[8] `http://tracker.virusshare.com:6969/`

4.4.1 Baseline scan for malicious binaries

Before scanning each of the known malicious binaries a baseline record is required. This demonstrates the effectiveness of the evasions. The first piece of malware analyzed is a generic trojan called Win32.Zapchast that has been in the wild since 2006. Owing to the age of this malware it has a fairly high detection ratio.

Table 4.10: Base Zpchast Scan

SHA256	83023e32bb12...
File name	VirusShare_7ed3caf5e5e9f276b72694b79ab17c90.sf.report
Generic Name	Trojan.Win32.Zapchast
Detection ratio	48/55

The second malicious binary scanned is the generic payload used by the Zeus malware to gain a foothold on a target's computer. As expected this has a fairly high detection ratio as well as it has been in circulation in the wild for a number of years.

Table 4.11: Base Zbot Scan

SHA256	5b9508b92a63...
File name	VirusShare_9fc364ccffe540c627733222641259c9.sf.report
Generic Name	Trojan.Zbot
Detection ratio	49/55

The next piece of malware analyzed is from the Sality family of malware. This malware family has also been in existence for a very long time with original detections recorded as early as 2003. As with the other pieces of malware, the detection rate on this malware is relatively high.

Table 4.12: Base Sality Scan

SHA256	742915067b83...
File name	VirusShare_e5120818378bd9b3766d45be99f96b59.sf.report
Generic Name	Win32/Sality.dropper
Detection ratio	45/55

Custom Malicious Application The previous applications were selected because they had been previously detected as malicious and as such would trigger an alert from an antivirus engine. It is important to build a custom application that performs what would

be considered malicious actions. An example of this would be a keylogger application. There are few instances where a keylogger application can be considered as non-malicious and in these cases exceptions can be added to the antivirus engine to ignore the application. In the event that a legitimate application needs to hook access to the keyboard, there is rarely the need to write to file after a keyboard event is triggered. These two items - in that sequence - exhibit the signature of a keylogger application. This section will cover the building of a keylogger which should be detected as malicious by an antivirus engine. While the end goal is for the application to trigger an antivirus alert, this may not be the end result which would necessitate using a prepackaged keylogger which already triggers and alert. The expected result from this test is not to be detected as malicious, but for antivirus engines (which claim to use heuristic detection) to trigger an alert.

Application Details Since the testing is targeting the Windows platform, the keylogger will be built using the C# programming language. It will make use of the Windows hooks exposed by the Windows API. The application will also try to hide itself from the user interface so that there is no way to exit the application aside from finding it and manually killing the process.

Build Process When building a keylogger, the following process will be adhered to. The first step of the application will hook the windows keyboard API. Hooking the Windows API is achieved by calling the "SetWindowsHookEx" function with a callback to a function that will execute the users code. In our application the callback chaining was implemented to ensure that any subsequent applications will also receive notification of a hooked event if they have setup a callback.

The actual callback in our application will only fire on a key-down keyboard event. Once an event fires that it has the key-down event, the code will open a file stream to a file called log.txt in the current folder and write the key-code to the file. Thereafter, the file stream will be closed and the "CallNextHookEx" function will be called which will continue the rest of the callback chain.

Once the callback hook is in place and our function is set up to receive the calls, the application will finally call the "ShowWindow" function with the SW_HIDE parameter. This will hide the application from the user's Windows interface and prevent a user from exiting the application.

Scan results After scanning the application for the first time with VirusTotal, the results are shown in Table 4.13. The results illustrate that none of the antivirus applications, barring the Norman antivirus, detected the keylogger as malicious (detected as "Obfuscated.AI!genr").

This most likely means that by building a custom application an attacker could potentially bypass an antivirus application without much hindrance.

Table 4.13: Keylogger Scan Results

SHA256	21f97a50f7ef...
File name	KeyLogger.exe
Detection ratio	1 / 53

4.5 Testing Process Usage

Having analyzed the different binaries for use when testing the evasion techniques, the following order will be employed when testing the evasion techniques in the Chapters 5, 6 and 7. While this is test order, there are cases when a packer or encryptor will refuse to work with a selected binary and as such will not be used in the tests.

Table 4.14: Antivirus Binary Test Order

Test Order	**Binary**	**Expected Outcome**
1	CMD Benign	Benign
2	Metasploit benign	malicious/warnings
3	Metasploit opcode	malicious/warnings
4	ZapChast	malicious
5	Zeus	malicious
6	Sality	malicious

4.6 Summary

This chapter demonstrated how the detection engine of an antivirus application will be tested. As this is a blackbox test, it is not possible to extract further details regarding the techniques used by the antivirus engines and as such, the testing needs to be as generic as possible. This caters for multiple antivirus engines. Having demonstrated how to control the antivirus applications en mass and test their internal detection engines, the next chapter will begin by testing the packer evasion technique.

Chapter 5

Evasion: Packers

5.1 Introduction

Having demonstrated how to set up and test the antivirus applications and their internal scanning engines, this chapter will begin with compiling these processes and performing the first set of tests using the packer evasion technique. Once the tests are completed, the results will be compiled and analysed at the end of the chapter.

5.2 Hypothesis

It is expected that the antivirus application tested will be looking for specific signatures in the form of either a sequence of bytes or an application hash. For this reason, packing the baseline application will change the baseline application's signature. This should, in turn, result in a lower detection rate. It is expected that detections that are made will be due to the signatures of the packer applications and not the signature of the baseline application. This detection of the packers' stubs does not mean that packers are malicious by nature, rather: the antivirus industry tends to mark tools that are used frequently by malware authors as malicious even though they have legitimate uses.

5.2.1 Goals

The results obtained in testing demonstrate that an average reduction in the number of detections of at least 10 percentage points can be gained against the antivirus engines that detect the baseline tests as malicious. Further, detections recorded in the test results belong to signatures generated by the packer and is not the signature of the baseline application. This will be demonstrated by the use of a benign application which will produce no malicious detections when submitted prior to packing. The benign application

will then be detected as malicious after being packed and submitted for testing.

5.3 Tests

The testing will be performed in two phases. The first set of tests will examine existing packers that exist in the wild and are publicly accessible. This will provide us with a base from which our custom packer can be compared. The second phase, will be to test a custom implementation of a packer tool - as discussed in the section 5.8. The creation of a custom implementation will allow us to compare the results of using a packer that might be already known to antivirus engines versus a completely new method. While testing the existing packers that are already in the wild, we will only be testing the three of the available packers mentioned by Roundy and Miller (2013). While it would be beneficial to test all the packers, 7 of the 10 packers that they name, are not available publicly or executable on a modern operating system in the way that they were originally built and meant to be executed. The testing section will also be split into multiple sections, the first section titled Existing Packer Tests will cover the scans with the existing packers. The next section Custom Packer Tests, will cover the building, as well as testing, of the custom tests.

5.4 Existing Packers Tests

The testing of the packers will begin with a number of known packers. These packers will allow us to record and compare the results from the antivirus scans of the custom malware that will be built and tested in the next section. By submitting the resultant binaries from these packers for scanning, we will be able to determine if the antivirus applications are detecting the packers signatures or if they are triggering detection based on the malicious content within. The packers that will be tested are listed in Table 5.1. These packers were selected as they are the only available packers that run on modern operating systems. Furthermore, these are the only packers that can still be obtained for testing purposes. Each of the packers will be tested with the tests listed in section 4.5. The tests will be grouped per packer.

Table 5.1: Packers Tested

Packer Name	**Antivirus Version**	**URL**
UPX	(v3.91)	`http://upx.sourceforge.net/`
ASPack	(v2.29)	`http://www.aspack.com/`
PECompact	(v3.02.2)	`https://bitsum.com/pecompact/`

5.5 UPX Background

The first packer to be tested is the UPX (Ultimate Packer for Executables) packer. UPX is a common packer that has been in the wild since the early 1990s (first released in 1998). While the packer was not written with a malicious intent, it became a major player in the malware community owing to its efficiency as well the extremely small code size required to unpack a binary packed by the UPX packer.

5.5.1 Benign Test With UPX

To begin the testing for UPX, a baseline scan will be performed with the benign binary as indicated in Chapter 4. After performing the scan, the results can be found in Table 6 and followed up with the detailed results shown in Table 6. These results show that only two antivirus engines detected the binary as malicious. Further: they showed the signatures indicating a trojan and not a signature related to UPX.

5.5.2 Metasploit Basic Scan Wrapped With UPX

The basic Metasploit template scan aims to test whether an antivirus application can detect the Metasploit template that is wrapped with the packer - or failing that detection, it can detect the application once it is executed. The UPX packed binary is generated with maximum compression using the following command:

```
.\upx.exe -f ..\plain\template_x86_windows.exe \
           -o ..\plain\template_x86_windows_upx.exe
```

As can be seen in Table 5.2, there is a decrease of 22 percentage points in the number of detections for the baseline binary when using UPX for the initial packing test.

Table 5.2: Comparison Against Original Baseline Detection Rates

Detection Rate		
Original	**Packed**	**Percentage Point Decrease**
22 / 55	10/55	22 pp

5.5.3 Metasploit Opcode Scan Wrapped With UPX

The opcode scan refers to executing the tests with a template binary that has machine code embedded within to execute an application by executing the dynamic code stored inside the template application. By executing the binary in this manner, we are technically giving it heuristics that would make the binary seem more malicious.

```
.\upx.exe -f ..\op\template_x86_windows.exe \
         -o ..\op\template_x86_windows_upx.exe
```

Table 5.3: Comparison Against Original Baseline Detection Rates

Detection Rate		
Original	**Packed**	**Percentage Point Increase**
29 / 55	37/55	15 pp

As can be seen in Table 5.3, there is an increase of 15 percentage points in the number of detections for the baseline binary when using UPX for the initial packing test. A full listing of the antivirus engines that detected the binary as malicious can be found in Table 1. The exact reason for this increase cannot be stated since as the antivirus companies do not explain their scanning processes. While this holds true, it is known that between the previous test and the current test, the only change was the inclusion of the opcodes which launched a non-malicious application (the calculator application). It stands to reason, therefore, that it is likely these opcodes are what the antivirus engines are detecting.

5.5.4 Malicious Binaries wrapped with UPX

The last set of tests for the UPX packer are the wrapping of the malicious binaries as defined in section 4.5. These binaries were packed with the default UPX settings and then submitted for scanning. A summary of the scan results can be found in Table 5.4. From this data, we can extrapolate an average reduction in the number of detections of 39.5 percentage points, for the malicious binaries when packed with UPX. While this is indeed useful when evading antivirus engines, it is by no means significant as more than half of the original antivirus engines that were tested will still detect the binary as malicious.

Table 5.4: UPX comparison of Packed vs Original Detection Rates

	Detection Rate		
Binary	**Packed**	**Original**	**Percentage Point Decrease**
Malicious Binary 1	29/55	48/55	35 pp
Malicious Binary 2	28/55	52/55	44 pp
Malicious Binary 3	N/A	52/55	N/A

Malicious Binary 3:

There are no scan results for the third malicious binary as UPX was not able to pack the binary. The inability to pack a binary takes place when the binary implements

some custom protections or is already compressed. When checking the binary for existing compressions of protection schemes with Protection ID[1] (Protection ID is a tool used to detect protection schemes employed by a binary), none were found. While this does not conclusively mean it is not already packed or using a known protection scheme, it instead means that the binary has been modified in such a way that these protection schemes are not easily identified.

Table 5.5: Common antivirus engines across all three malicious binaries

AntiVir	Microsoft
Avast	Norman
Comodo	Qihoo-360
DrWeb	Sophos
ESET-NOD32	Symantec
Fortinet	TrendMicro
Ikarus	TrendMicro-HouseCall
Kaspersky	VBA32

Furthermore, as can be extracted from the data in the Table 5.5, only 16 of the 55 anti-virus engines were able to detect all three malicious binaries. This is a cause for concern since all three of the binaries have been circulation for at least one year (the last submission time stamp indicates 2012-11-27).

5.6 ASPack Background

ASPack is a commercial packer that gained use in the malware community due to its effectiveness to compress 64 bit applications which a number of other packers had issues packing. ASPack had roughly only 1.3% of the malware market with regards to packing (Roundy and Miller, 2013). It is one of the few packers that are still publicly accessible and executable on modern operating systems.

5.6.1 Benign Test With ASPack

To begin the testing for ASPack, a baseline scan will be performed with the benign binary as indicated in Chapter 4. After performing the scan, the results can be found in Table 8 and followed up with the detailed results shown in Table 9. These results show that only two antivirus engines detected the binary as malicious. Further they showed the signatures

[1]http://pid.gamecopyworld.com/

indicating a joke application and a generic malicious application, and not any signature related to ASPack. It should be noted that the antivirus engines that detected the binary as malicious are not the same as those that detected the UPX binary as malicious.

5.6.2 ASPack Metasploit Basic Scan

Since the ASPack application is driven by a graphical user interface, we cannot record the commands required to pack the binary similar to the way the UPX commands were recorded. To pack the Metasploit binary, simply select the application from within the ASPack application. This will immediately compress and make a backup of the file. Once the application is packed and submitted to VirusTotal, we get the following results. These results will be analysed further in section 5.9.1.

Table 5.6: ASPack Basic Scan

SHA256	6312d0dabe92...
File name	template_x86_windows.exe
Detection ratio	6 / 55

Table 5.7: ASPack Basic Scan

Antivirus	**Version**	**Common Name**
CMC	1.1.0.977	Hoax.Win32.BadJoke.ScreenFlicker!
Malwarebytes	1.75.0.1	Backdoor.Bot.gen
K7anti-virus	9.183.13139	Backdoor (04c5312d1)
K7GW	9.183.13139	Backdoor (04c5312d1)
AhnLab-V3	2014.08.24.00	Backdoor/Win32.Bifrose
Ikarus	T3.1.7.5.0	Trojan.Win32.Swrort

Table 5.8: Comparison Against Original Baseline Detection Rates

Detection Rate		
Original	**Packed**	**Percentage Point Decrease**
22 / 55	6/55	29 pp

As can be seen in Table 5.8, there is a decrease of 29 percentage points in the number of detections from the baseline binary when using ASPack for the initial packing test. It should be noted that this reduction in detection rates is most likely due to no exploitable opcodes being embedded in the binary.

5.6.3 ASPack Metasploit Opcode Scan

As with the previous test: we pack the Metasploit binary with the opcodes embedded with ASPack and then submit the compressed application to VirusTotal. Once submitted we get the following results. These results will be analysed further in section 5.9.1.

Table 5.9: ASPack Opcode Scan Summary

SHA256	f1b93dddad1c...
File name	template_x86_windows_op.exe
Detection ratio	29 / 55

As can be seen in Table 5.11, there is a decrease of 35 percentage points in the number of detections for the baseline binary when using ASPack. As observed with the UPX tests, once the baseline binary is embedded with opcodes that are associated with an exploit the detection rates do increase. While the detection rates did not increase significantly (four more than the previous test), it still indicates that certain scanners are looking for specific signatures which are represented via the opcodes for detection. The common antivirus engines between the three malicious binaries can be found in table 5.10.

Table 5.10: ASPack AV Scan Results

Antivirus	**Version**	**Common Name**
CMC	1.1.0.977	Hoax.Win32.BadJoke.ScreenFlicker!O
Malwarebytes	1.75.0.1	Spyware.Passwords
TrendMicro-HouseCall	9.700.0.1001	PAK_Generic.009
Kaspersky	12.0.0.1225	HEUR:Trojan.Win32.Generic
NANO-anti-virus	0.28.2.61721	Trojan.Win32.Inject1.cugnjk
DrWeb	7.0.9.4080	Trojan.Inject1.33413
TrendMicro	9.740.0.1012	PAK_Generic.009
Microsoft	1.10904	Trojan:Win32/Swrort.A
Ikarus	T3.1.7.5.0	Trojan.Win32.Swrort
AVG	14.0.0.4007	Crypt3.IJL

Table 5.11: Comparison Against Original Baseline Detection Rates

Detection Rate		
Original	**Packed**	**Percentage Point Decrease**
29 / 55	10/55	35 pp

5.6.4 ASPack Malicious Binary Scan

The results from packing the malicious binaries can be found in Table 2. When comparing the results from packing the malicious binaries with ASPack to the original detection rates from scanning the unpacked binaries, the following results are listed in Table 5.12. An average decrease of 60 percentage points was observed in the number of detections between the original and packed version for each of the three binaries.

Table 5.12: ASPack comparison of Packed vs Original Detection Rates

	Detection Rate		
Binary	**Packed**	**Original**	**Percentage Point Decrease**
Malicious Binary 1	16/55	48/55	58 pp
Malicious Binary 2	18/55	52/55	62 pp
Malicious Binary 3	19/55	52/55	60 pp

Only five antivirus engines were able to detect all three of the packed binaries as malicious; namely AntiVir, Avast, Comodo, ESET-NOD32 and Kaspersky.

5.7 PECompact Background

PECompact is also a commercial packer similar to that of ASPack. The packer is one of the top five packers with regards to packers employed amongst the malware community with an estimated market share of 2.6% at the time of the publication (Roundy and Miller, 2013). As with ASPack, the packer gained a large market share owing to its effectiveness in both compression and application control flow protection.

5.7.1 Benign Test With PECompact

To begin the testing for PECompact, a baseline scan will be performed with the benign binary as indicated in Chapter 4. After performing the scan the results can be found in Table 10 and followed up with the detailed results shown in Table 11. These results show that nine antivirus engines detected the binary as malicious. This was significantly more than any of the previous packers. The antivirus listing showed that most of the antivirus applications detected the binary as a generic malicious application or as a Trojan. Only the antivirus Agnitum detected the binary with a PECompact signature.

5.7.2 PECompact Metasploit Basic Scan

In the same manner as ASPack, there are no automated commands to pack an application and packing is handled via a graphical user interface. PECompact supports a number of compression and protection schemes for protecting an application. All tests will be performed with the default settings which uses a medium level of compression and no anti-reversing schemes. After packing the application and submitting it to VirusTotal, the results found in Table 5.13 (below), will be analysed in section 5.9.1

Table 5.13: PECompact Basic Scan Summary

SHA256	1f83dce7884b...
File name	template_x86_windows.exe - PECompact.exe
Detection ratio	11 / 55
Baseline Detection ratio	22 / 55

A decrease of 20 percentage points was observed in the number of detections when compared to the baseline binary. The only observation that could be extracted from the current test was that PEcompact has a low detection rate for the stub that it uses for extraction. Unfortunately, this specific stub cannot be easily extracted and scanned alone in order to determine if this is the case.

Table 5.14: PECompact basic scan results

Antivirus Engine	**Version**	**Detected As**
Bkav	1.3.0.4959	HW32.CDB.D7c0
Malwarebytes	1.75.0.1	Backdoor.Bot.gen
Agnitum	5.5.1.3	Packed/PECompact
F-Prot	4.7.1.166	W32/Threat-HLLIP.gen!Eldorado
Symantec	20141.1.0.330	Suspicious.Emit
TrendMicro-HouseCall	9.700.0.1001	PAK_Generic.001
TrendMicro	9.740.0.1012	PAK_Generic.001
AhnLab-V3	2014.08.25.00	Dropper/Win32.OnlineGameHack
Commtouch	5.4.1.7	W32/Threat-HLLIP-based!Maximus
Ikarus	T3.1.7.5.0	Virus.Win32.Heur
Qihoo-360	1.0.0.1015	Malware.QVM17.Gen

5.7.3 PECompact Metasploit Opcode Scan

The last of the existing packer tests that will be performed is the opcode binary packed with PECompact. The same settings as those used in the previous basic Metasploit scan will be used. The results from this scan be will be analysed in section 5.9.1.

Table 5.15: PECompact Opcode Scan Summary

SHA256	4d968080fb3a...
File name	template_x86_windows_op.exe - PECompact.exe
Detection ratio	15 / 55
Baseline Detection ratio	22 / 55

As with the previous test, the scan results are significantly lower to any of the other packers with a decrease of 15 percentage points was observed when compared to the baseline binaries scan. While there is a decrease when compared to the baseline binary, there is an increase in the number of detections of 5 percentage points when compared to the previous test. As with the previous test, this is most likely due to the inclusion of the OP codes which is the only differentiator from the previous test.

Table 5.16: PECompact Opcode Scan Antivirus Results

Antivirus Engine	**Ver**sion	**Detected As**
CMC	1.1.0.977	P2P-Worm.Win32.SpyBot!O
Agnitum	5.5.1.3	Packed/PECompact
TrendMicro-HouseCall	9.700.0.1001	PAK_Generic.001
Kaspersky	12.0.0.1225	HEUR:Trojan.Win32.Generic
NANO-antivirus	0.28.2.61721	Trojan.Win32.Inject1.cugnjk
Rising	25.0.0.11	PE:HackTool.Swrort!1.6477
DrWeb	7.0.9.4080	Trojan.Inject1.33413
TrendMicro	9.740.0.1012	PAK_Generic.001
Microsoft	1.10904	Trojan:Win32/Swrort.A
Ikarus	T3.1.7.5.0	Trojan.Win32.Swrort
AVG	14.0.0.4007	Crypt3.IJL

5.7.4 PECompact Malicious Binary Scan

The base results from packing the malicious binaries with PEComact can be found in Table 3. When comparing the results from packing the malicious binaries with PECompact to the original detection rates from scanning the unpacked binaries, the following results are

listed in Table 5.12 are found. An average decrease of 59 percentage points was observed in the number of detections across the three malicious binaries when compared to their baseline scans.

Table 5.17: PECompact comparison of Packed vs Original Detection Rates

	Detection Rate		
Binary	**Packed**	**Original**	**Percentage Point Decrease**
Malicious Binary 1	18/55	48/55	55 pp
Malicious Binary 2	18/55	52/55	62 pp
Malicious Binary 3	19/55	52/55	60 pp

While not listed above comprehensively, it can be found that only six antivirus engines were able to detect all three of the packed binaries as malicious. These antivirus engines are listed in Table 5.18.

Table 5.18: Common antivirus engines between Malicious binary 1,2,3 with PEcompact

Agnitum
AntiVir
Avast
ESET-NOD32
TrendMicro
TrendMicro-HouseCall

5.8 Custom Packers Tests

The building of a packer is not clearly defined in previously conducted research and therefore the following method for building a packer is acquired from the analysis of existing malware. When analysed, the following methodology was revealed. A packer is generally comprised of two parts. The dropper which is generated by the packer application, is what gets distributed to the end user. This application handles the evasion of the antivirus application as well the payload delivery - which is usually contained within the application. To build a custom packer the following high level strategy can be implemented by a malware author. The author first selects a method for compressing the binary. This may be as simple as the zip or a custom implementation. Once this is selected, an application is built that is known as the packer. This will accept a binary as an input, compress it and then inject it into the dropper application. The selection of a compression mechanism is optional but is highly recommended as it provides another

level of obfuscation. The dropper application can be built in two ways. The dropper will be built as a distinct and separate binary which will manage the extraction and execution of the packed code. The alternative and more complicated method is to build a code stub which is injected along with the packed binary into any benign application that may already exist on an end user's system. This second method is more effective since the application may already be white-listed for execution on the end user's system and does not provide as easy a signature to detect. Depending on the type of dropper that the attacker wishes to use, the packer will need to be built to either: generate a dropper application or just a code stub that will be injected into the transport application.

Table 5.19: Dropper Type Pros vs Cons

Dropper Type	**Pro**	**Con**
Static Binary	Simpler To Build	Static Binary Easily Detected
Dynamic Binary	Dynamic Binary Harder To Detect	Complicated Build Process

With the aforementioned knowledge in mind, it was decided that the initial type of dropper was selected for use (i.e. a static binary that had data only injected). This method of packing was selected because it is both simpler and less prone to error. The stub and data injection is significantly more complicated as it requires rewriting different parts (the original entry point) of the binary which is required to access the injected stub. Furthermore, the benefits of the stub injection (which is a dynamic binary) only assists evasion when used as part of a self-replicating package which is required by either worms or viruses. Since the binaries that are submitted for testing have to be submitted manually, changing the dropper executable is simply achieved by using different open source projects while adding the required extraction code.

5.8.1 Implementing the custom packer

The tools for building a packer application can be built using high level languages such as C# or Python or another scripting language as is common among certain malware authors (Borello *et al.*, 2010). The alternative is to build the application in a lower level systems level type language such as C. The trade-off between the choice of languages comes down to how fast the author can implement the packer in a given language. Higher level scripting languages that are able to compile to executable binary applications are the most likely to be selected as it means the runtime requirements on the end user's system are lower.

In the same chain of reasoning, lower level languages provide fairly independent runtime requirements by default, but at the cost of reimplementing a number of simple functions provided by higher level functions. To speed up the build process, it was determined that a higher level language would be used. The higher level language that was selected was the C# language which runs on the Microsoft .Net framework. There is no specific reason for using C# other than ease of use and portability. The same application packer could be built by another rapid application development language and will be left for future work. By building the packer using C# and using the packer / dropper method of building a packer, there is a decreased chance of building the packer incorrectly. The dropper application will be implemented by building a template binary that has a fixed resource signature. This signature will then be found when the packer is executed and replaced with the packed content.

5.8.2 Dropper Tests

As explained in section 2.9.4, packers are composed of two parts. For the dropper tests, the part of the application that is considered the packer will not be tested as there is nothing that can be considered malicious in the packer application. The tests described below will be for the dropper that will be executed on an end user's system. The structure of the tests below will be executed in the same order as listed in Table 4.14. The dropper that will be used for all the tests is generated by running the packer binary while using each of the binaries listed in Table 4.14 as the source. The output from this application is what will be submitted for final testing.

5.8.3 Test with benign application

The testing will begin with a benign application, this is used to determine if the antivirus application is detecting the malicious code or the evasion wrapper. Once the benign binary is packed and submitted to VirusTotal, the results in Table 5.20 are noted. These results indicate (excluding one antivirus) that none of the other applications detected the binary as malicious.

Table 5.20: Benign Application Scan Results

SHA256	ea39f07cf220...
File name	demo.exe
Detection ratio	1 / 49

Table 5.21: Benign Application AV Scan Results

Antivirus	**Result**
CMC	Hoax.Win32.BadJoke.ScreenFlicker!O

5.8.4 Final packer test with Metasploit binary

As with the previous test, we prepare the dropper for upload but instead of using the generic calculator application, we use the Metasploit testing binary. The scan results from uploading this binary can be found below. Note: the Metasploit that is packed does not contain any exploit opcodes. The results from packing the malicious binaries with the custom packer can be found in Table 4.

Table 5.22: Metasploit Scan Details

SHA256	df06d7bc21f6...
File name	SelfExtractor.exe
Detection ratio	2 / 49

Table 5.23: Metasploit AV Detection Test Results

Anti-virus	**Result**
Avast	Win32:Malware-gen
Malwarebytes	Backdoor.Bot.gen

5.9 Reports

As mentioned previously in subsection 4.3.2, the eicar file could not be used for any of the tests and as a result it is not covered in further detail for the rest of the section

5.9.1 Existing Packer Reports

UPX Metasploit Basic Test Results: Looking at the tabulated results in Table 25 the basic Metasploit scan, while being wrapped with UPX, had an increased detection rate from the original detection rate of six for the gcc test. When looking at the tests for the pre-compiled binary we do gain a small decrease in the detection rate, though this is attributed to the small number of scanners detecting the packers signatures instead of the enclosed payload.

UPX Metasploit opcode Test Results: In Table 26 we see a significant increase in detection by antivirus engines. The majority of the antivirus engines seem to detect the occurrence of malicious behaviour and flag the binary as a generic malicious binary. The behaviour that is detected according to the F-Secure antivirus index[2] is that of a generic Trojan application. This labeling of a Trojan does not conclusively point to the antivirus application detecting the internal payload since packer applications are generally associated with Trojans.

ASPack Metasploit Basic Code Test Results: Detection ratio: 6 / 55
When being wrapped with the ASPack packer and submitting the application to VirusTotal we see a minimal detection ratio. This is most likely due to the payload being benign and the antivirus applications simply detecting packers stub application.

ASPack Metasploit Opcode Test Results: Detection ratio: 10 / 55
When the Metasploit binary containing the opcodes to execute an application is wrapped with the ASPack packer. The application submitted to VirusTotal presents a slight increase in detections compared to the non opcode embedded version. This increased detection is possibly due to the sample application launching the calculator application which is considered a test case of many malicious pieces of malware.

PECompact Metasploit Basic Code Test Results: Detection ratio: 11 / 55
The metasploit binary that is packed using the PECompact application has a higher detection rate than the ASPack packer. Since we are looking at the basic payload which does not launch any application, the antivirus engines are once again showing signatures for generic Trojans and backdoors. This is an indication that the antivirus engines are detecting the packers signature and not the internal payload.

PECompact Metasploit Opcode Test Results: Detection ratio: 11 / 55
The Metasploit binary with the included payload did not result in an increase in detection rate. This further confirms that the antivirus engines were simply detecting the packers signature that is emitted by default when packing an application.

5.9.2 Custom Packer Reports

Benign Test Results As expected none of the antivirus engines detect the wrapper application as malicious (only one engine flags the application as a joke application). The joke detection is possibly because the application launched is the calculator application.

[2] `http://www.f-secure.com/v-descs/trojan_w32_graftor.shtml`

Metasploit Test Results From the results we can see the only two antivirus engines detect the packed binary as malicious. It maybe be possible that the payload will be detected at a later stage. Getting the payload past the first level of scanning is a significant win.

5.10 Summary

As mentioned previously in section 2.5, packers were one of the first methods used to evade antivirus engines. Consequently, it is the first method of evasion that was tested. After using the existing packers to pack the test binaries, the results were submitted to VirusTotal for scanning and the following results were found and compiled into Table 5.24. The results in table 5.24 represent the final score from virus total indicating the number of antivirus engines that detected the binary as malicious, while the percentage point difference compared to the original baseline binary is shown in brackets next to the scan results. These percentage points indicate whether there was an increase or decrease in the number of detections, with negative values indicating an increase in the number of detections, while positive numbers indicate a decrease in the number of detections.

Table 5.24: Detection Rates Side By Side Summary

	Detection Rate (Percentage Points)				
Binary	**UPX**	**ASPack**	**PECompact**	**Custom Packer**	**PP Average**
Metasploit Base	10/55 (22 pp)	6/55 (29 pp)	11/55 (20 pp)	2/55 (36 pp)	27 pp
Metasploit Opcode	39/55 (-29 pp)	29/55 (-11 pp)	15/55 (15 pp)	22/55 (2 pp)	-6 pp
Malicious Binary 1	29/55 (35 pp)	16/55 (58 pp)	18/55 (55 pp)	13/55 (64 pp)	53 pp
Malicious Binary 2	28/55 (44 pp)	18/55 (62 pp)	18/55 (62 pp)	14/55 (69 pp)	59 pp
Malicious Binary 3	0/55 (0 pp)	19/55 (60 pp)	19/55 (60 pp)	18/55 (62 pp)	46 pp
Averages	14 pp	40 pp	42 pp	47 pp	36 pp

There are a number of points that can be extracted from the Table 5.24. Working through the results from the top of the table: none of the packers were able to process the eicar file except for the custom packer which still had a zero detection ratio. The reason for none of the packers being able to process the file is because they require a valid PE binary to modify. Since the eicar file is just a text file that works as a COM file as well, it cannot actually be modified. The benign binary was tested with each packer to establish if the antivirus engines were detecting the malicious code or the wrapper code. Following the benign tests, the Metasploit base file was tested. This is the original baseline file that was selected in section 4 after a number of other options were originally considered and subsequently eliminated as options. The baseline binary started with a base detection

rate of 22/55. This detection rate either dropped or increased when different packers were independently applied. On average the number of detections dropped by 27 percentage points across all the packers. While there was a large drop in this initial scan, it was attributed to the significant change of the baseline binary coupled with the fact that it had no malicious code embedded in it, indicating most antivirus engines ignored the binary.

To test if adding malicious code to the application would change the detection rates, the next test added exploit opcodes to launch a benign application in a non-conventional manner. As mentioned previously, the next step was to add some additional code to the template binary to make it seem more malicious to the antivirus applications. After adding the code and scanning it, the new template binary had a total detection rate of 23/55 with only one more antivirus engine detecting the binary as malicious. While only one antivirus engine detected the new template binary as malicious, after packing the template binary with each packer and then submitting it for testing a new pattern is discovered. For UPX and ASPack the detection rate increased instead of decreasing as expected, with an average increase of 20 percentage points in the number of detections across the two packers. While the detection rates for the custom packer only got reduced by two percentage points, only PEcompact gained a significant reduction in the number of detections with a decrease of 15 percentage points.

A final test was performed to determine the effectiveness of working with a baseline binary. This final test was to use all the packers to pack a set of known malicious binaries and compare the results. This final test compromises rows three, four and five in Table 5.24 for the three selected malicious binaries indicated in section4.4. Binary number one had an average reduction in the number of detections of 53 percentage points across the four packers. The average reduction in number of detections for binary number two was increased to 59 percentage points with binary number three having a reduction of 46 percentage points in the number of detections across the three packers that were able to pack it (UPX was not able to pack the binary). This resulted in an overall reduction of 52 percentage points in the number of detections across the three binaries.

The custom packer performed marginally well by gaining the lowest detection rate of all the packers for each of the scans. The packer could have been built differently to achieve better results, but the aim of using the custom packer was not to gain a perfect bypass, but rather to show that it could perform better than existing packers with minimal work involved. For malware authors this is good news as it means that building a custom packer as part of the release plan, will result in the malware being able to evade the vast majority

of the antivirus engines employed. Based on the results in Table 5.24, malware authors that do not have the required development skills can still use existing packers to gain some reduction in the detection of the malware they deploy. In conclusion, this chapter dealt with the initial set of tests and demonstrated how the tests should be performed as well as their effectiveness against the antivirus engines available at the time. The next chapter will discuss the encryption evasion technique and demonstrate the effectiveness of encrypting a malicious binary to evade modern antivirus applications.

Chapter 6

Evasion : Encrypters

6.1 Introduction

Similar to the work described in chapter 5, this chapter will perform a set of tests to gauge the effectiveness of modern antivirus applications in detecting a set of protocol binaries wrapped in the encryption evasion technique as introduced in section 2.6. This chapter will use the processes and methodologies listed in Chapters 3 and 4 to test the effectiveness of the encryption evasion technique.

6.2 Hypothesis

It is expected that the antivirus application tested will be looking for specific signatures and therefore, encrypting the baseline application will change the application's signature. This should, in turn, result in a lower detection rate. It is expected that the detections raised by the antivirus application will be due to the signatures of the encryption stubbs not the signature of the baseline application. Based on the results from Chapter 5, this chapter will attempt to match the reduction gained by the packers or gain an increase in the reduction. Further, detections recorded in the test results belong to signatures generated by the packer and is not the signature of the baseline application. This will be demonstrated by the use of a benign application which produces no malicious detections when submitted prior to encryption. The benign application will then be detected as malicious after being encrypted and submitted for testing.

6.2.1 Goals

The results obtained from testing in the previous chapter demonstrate that an average reduction in the number of detections of at least 10 percentage points can be gained

against the antivirus engines that detect the baseline tests as malicious. As such the goal for this chapter will also attempt to achieve an average reduction of at least 10 percentage points.

6.3 Tests

The testing will be performed in two phases. The first phase of testing will test a set of existing encryptors that exist in the wild and are publicly accessible. This will provide us with a base from which our custom encrypter can be compared. The second phase will be to test a custom implementation of an encrypter tool as discussed in the section 5.8. The creation of a custom implementation will allow us to compare the results of using a packer that generates a known signature as part of its decryption routine which may be already known to antivirus engines versus a custom application that would not have a recorded signature. This in turn can be used to determine if the antivirus applications are looking for signatures in the encrypting applications or if they are looking for signatures in the malicious binaries or if the antivirus applications are using heuristics to perform their detection.

As for the tests that will be executed and the order of the tests, they are listed in the Table 4.5. The order and tests are the same as those executed for Chapter 5.8. A benign application will first be tested to determine if the antivirus applications are scanning for signatures in the decryption routines. Once the benign test is completed, the baseline test will be executed using the baseline application selected in the section 4 on baseline binary selection.

6.4 Existing Encrypters

For the baseline testing of the existing encryption application, two encryptors that are available and run on modern operating systems will be selected for use.. The two applications that were selected are Hyperion[1] and PEScrambler[2]. The two applications provide the functionality required to be tested - which is to wrap a binary by encrypting its content and then appending a decryption stub to the encrypted binary so that the binary can be decrypted on execution.

[1] `http://nullsecurity.net/tools/binary.html`

[2] `https://code.google.com/p/pescrambler/`

Table 6.1: Encrypters to be tested

Encypter	**Version**
PEScrambler	v0.1
Hyperion	v1.0

6.5 Hyperion Background

The Hyperion packer[3] is the first of the encryptors to be tested. The encryptor was selected because it is one of the few remaining publicly available encryptors that work on a modern operating system without the need for code modification. As mentioned in the introduction, the tests will start with a benign test and then move on to the baseline test and finally a malicious test.

6.5.1 Benign Test

The first test is to set the baseline for the future tests. It will allow for verification if an antivirus engine detects a binary as malicious purely based on a byte signature in the decryption stub. As discussed in 4.3.1, the application that was selected was the 32bit version of cmd.exe as the application has no known side effects and is executable on all modern operating systems (without the need for dedicated dependencies). The baseline scan for cmd.exe is found in the Table 4.2 in the section 4.3.1 and as expected is not reported as malicious at all.

After encrypting the application and then submitting it for testing, the results as listed in Table 6.2 show a detection ratio of 20/55. From this it can determined that the antivirus engines that are detecting this binary as malicious are simply scanning for a known byte signature that would be generated by the Hyperion Application.

Table 6.2: Hyperion Benign Scan

SHA256	f70cebe27268...
File name	cmdhp.exe
Detection ratio	20/55

6.5.2 Baseline Test

The following section covers process of encrypting and then scanning the baseline binary with Hyperion. The scans for the baseline binary can be found in table 6.3 which contains

[3] `http://nullsecurity.net/tools/binary.html`

no exploit code, this is followed by a scan of the baseline binary with exploit embedded as seen in table 6.4.

Table 6.3: Baseline Encrypted Scan with Hyperion

SHA256	a3080b3c97d9...
File name	template_x86_windowshp.exe.report
Detection ratio	23/55

When compared to the previous test the results in Table 6.3 show only a small increase of 2 percentage points in the number of detections. This indicates that only three of the anti-virus engines are looking at the end result of the decryption process. The small increase could be attributed to the template being inert and as such not triggering any heuristics for the antivirus applications.

6.5.3 Hyperion Baseline Scan with Opodes

Table 6.4: Exploit Enabled Baseline Encrypted Scan with Hyperion

SHA256	900a0b4fcb6c...
File name	template_x86_windows_ophp.exe.report
Detection ratio	23/55

After submitting the template binary with the opcodes embedded for scanning, the results found in Table 6.4 indicate no increase in detection rates. The verification that the same file was not submitted for both tests can be achieved by comparing the sha256 hash values.

6.5.4 Malware Test

With Hyperion, only one sample of the three listed in the malware test (Table 4.9) was able to encrypt successfully. The sample that was able to encrypt successfully was Malicious Binary 2 (the Zeus Bot) and as a result it is the only malware sample tested in this section. Malicious Binary 2 :

Table 6.5: Zeus bot Malware Scan with Hyperion

SHA256	0c1e63a8d1a5...
File name	hp_VirusShare_9fc364ccffe540c627733222641259c9.sf.report
Detection ratio	26/54

When comparing the results from the malware submission in Table 6.5 it can be seen that there is a significant drop in detection rates of 47 percentage points when compared to the original scan results for the malicious binary. Unfortunately because there are so few malicious binaries that can be successfully encrypted with Hyperion it is difficult to extract a definitive comparison.

6.6 PEScambler Background

The final encryptor to be tested is the encryptor known as PEScrambler[4]. The original repository was available at[5], but this is no longer open to the public. As with the Hyperion tests, this section will begin with the benign test to set a baseline and determine if the antivirus engines have a known byte signature for the encryptors decryption stub.

6.6.1 Benign Test

Table 6.6: PEScrambler Benign Scan

SHA256	fb69c4d27df8...
File name	cmdpe.exe.sf.report
Detection ratio	7/55

While there is a slight increase in detection rates (seven antivirus engines detected the benign application as malicious), in comparison to the Hyperion test, the PEScrambler had a significantly lower detection ratio for the benign application (7/55 versus 20/55).

6.6.2 Baseline Test

The following section covers the process of encrypting and then scanning the encrypted binary with PEScambler. The scans for the baseline binary in Table 6.7 with no exploit code embedded is followed by a scan of the baseline binary with exploit embedded in Table 6.8.

After submitting the encrypted baseline template for scanning, the results in Table 6.7 indicate that there is a decrease of 18 percentage points in the number of detections when compared to the baseline template scans.

[4] `https://github.com/Veil-Framework/Veil-Evasion/tree/master/tools/pescrambler/`
[5] `https://code.google.com/p/pescrambler/`

Table 6.7: Baseline Encrypted Scan with PEScrambler

SHA256	aa694414c81c...
File name	template_x86_windowspe.exe.report
Detection ratio	12/55

6.6.3 PEScrambler Baseline Scan With Opcodes

Table 6.8: Exploit Enabled Baseline Encrypted Scan with PEScrambler

SHA256	59275c39364b...
File name	template_x86_windows_oppe.exe.report
Detection ratio	13/54

After submitting the template with the opcodes embedded for scanning, the results in Table 6.8 indicate a decrease of 18 percentage points in the detection rate when compared to the template opcode binaries original scan results.

6.6.4 Malware Test

The following section details the process of encrypting a live malware sample and then scanning the encrypted binary. The three samples below were encrypted successfully with PEScrambler. The application does not provide any information on why it is not able to encrypt an application and as a result, only the successful malware samples will be listed below. After submitting the encrypted copy of malicious binary 1 for scanning and analysing the results in Table 5, it is found that there is a decrease of 73 percentage points in the rate detection. This is a significant result in the decrease of detection rates and when compared to all other tests completed up this point, it is the highest decrease in detection.

After submitting the encrypted copy of malicious binary 2 for scanning and analysing the results in Table 5, it is found that there is a decrease of 56 percentage points in the rate of detection. While this decrease in rate of detection is noteworthy, it does not compare as favourably to the decrease in the detection rates for malicious binary 1. One possible explanation is that Zeus malware authors may have previously encrypted copies of the malware with PEScrambler.

After submitting the encrypted copy of malicious binary 3 for scanning and analysing the results in Table 5, it is found that there is a decrease in the detection rates of 78 percentage points. As with malicious binary 1, a high reduction in the detection rates

is once again observed and further points to the possibility that the Zeus scan was an oddity.

6.7 Custom Encryptor

The implementation of encryptors is very similar to that of packers. The Microsoft .Net Framework (.Net) is used and the encrypted data is stored in the dropper binary. In the same manner as the packer, the encryptor is built using the .Net platform with the C# programming language. Furthermore, since the encryptor works in a similar method to the packer, the majority of the packers code will be used and then expanded (where needed) to add the encryption layer. As with the packer, the encrypter is split into two parts, namely the encryptor and the dropper. Similarly to the packer, the dropper will store the payload as well as handle the decryption and final execution of the binary that was packed. The encryption used in the implementation is AES. Further, the payload is encrypted directly and is not zipped before encryption.

In the hypothesis it was stated that we would be testing the techniques used to evade an antivirus, instead of testing the specific implementations of particular techniques against modern antivirus engines. Consequently, a custom implementation of an encryptor will need to be built which will test the second of the two evasion techniques that was mentioned previously in 2.5.3.

The two basic techniques that were built and tested, are the packer and encryption techniques. These two methods are very similar because they are different stages of the evolution of basic evasion techniques. The testing of metamorphism as well as the other more advanced evasion techniques will need to be completed as a separate research undertaking owing to the depth and complexity of these techniques. The encryptor tests are similar to that of the packer with a slight change: instead of performing a dummy scan with no valid payload, testing will start directly with a payload that has some basic opcodes to launch the calculator application in Windows. As with the packer tests, the tests are not meant to evaluate the encryptor, rather to test the dropper application which contains the payload.

6.7.1 Baseline Test

When testing the encrypter with the basic template payload the results as found in table 6.9 were observed.

Table 6.9: Baseline Scan With Custom Encrypter

SHA256	4cce95b26fd4...
File name	SelfDecrypter.exe
Detection ratio	0 / 49

The scan shows that none of the antivirus engines detected the payload that was embedded in the binary as malicious. Furthermore the decryption stubb application was not detected as malicious either because it was a custom application.

6.7.2 Baseline With Opcodes Test

A final scan was done with a payload that executed basic opcodes in order to launch the Windows calculator application. This method of executing code is technically considered more malicious and is likely to trigger more alarms for the antivirus engine. The results from this scan be found in Table 6.10 and as with the previous test, none of the antivirus engines detected the binary as malicious.

Table 6.10: Baseline Scan With Opcodes

SHA256: 4aad495e7176...
File name: SelfDecrypter_OP.exe
Detection ratio: 0 / 49

6.8 Summary

After performing all the tests for the chapter, the results are shown in Table 6.11. The table indicates a baseline by which to compare the rest of the data when determining if the antivirus engines are detecting the encryptor's signature as malicious or if the actual embedded binary as malicious. Based on the hypothesis and observation in the previous results indicated in Table 5.24, it was expected that a number of antivirus applications would detect the decryption code as malicious. The results from the benign scan confirms this notion by indicationg that both Hyperion and PEScrambler had malicious detections when encrypting the previously clean benign application. Note: The benign scores not considered in averages for table 6.11 and is simply displayed for easier comparison.

Table 6.11: Encryption Comparison Summary

Binary	**Hyperion**	**PEScrambler**	**PP Average**
Benign	20/55 (-36 pp)	7/55 (-13 pp)	-25 pp
Metasploit	23/55 (-2 pp)	12/55 (18 pp)	8 pp
Metasploit OP	23/55 (0 pp)	13/55 (18 pp)	9 pp
Malicious Binary 1	N/A	8/55 (73 pp)	37 pp
Malicious Binary 2	26/55 (47 pp)	21/55 (56 pp)	52 pp
Malicious Binary 3	N/A	9/55 (78 pp)	39 pp
Percentage Point Average	9 pp	49 pp	29 pp

After the benign scan was completed, the next two scans performed were the Metasploit and Metasploit OP scans. These scans were meant to test an antivirus engines ability to detect the flagged binaries. After performing the scan with Hyperion, an increase of 2 percentage points were observed in the detection rates for the base Metasploit template binary. The PEScrambler showed a decrease of 18 percentage points in its detection rates. The increase in the detection rates of Hyperion is most likely due to the common use of Hyperion in a number of evasion frameworks - which have resulted in an increased detection rate, when coupled with the already malicious Metasploit template. For this reason, the increase in detection is reasonably expected.

The decrease in the detection rates is due, in part, to the opposite effect observed by the Hyperion scan. This effect is one in which a known encryptor increases the detection rate while an unknown encryptor decreases the detection rate. Even though PEScrambler is open source, it is not as popular as Hyperion when being used in the wild. For this reason, when encrypting an application it is able to provide evasion capabilities as expected.

When performing the scans with the OP code template binary the following results were observed. When scanned with Hyperion no change in percentage points were observed between the baseline OP binary and the encrypted binary. The same results as the original baseline scan were observed for the OP code binary with a decrease of 18 percentage points.

For the malicious binary tests, two of the binaries could not be encrypted with Hyperion. The reason why Hyperion could not encrypt these binaries is not provided by Hyperion. In the case of the one binary that was able to encrypt successfully, there was a decrease of 47 percentage points in detection rates. In the case of PEScrambler, all the binaries were able to encrypt successfully and resulted in an average reduction of 69 percentage points across the three malicious binaries tested.

There was an odd case observed with the tests for Malicious Binary 2 which showed a higher than normal detection rate for PEScramble when compared to the other binaries. It is assumed that this is because the binary that was tested has previously used this

specific encryptor in the wild, and a larger number of antivirus engines have signatures for its encrypted form. Overall the results for the malicious binary detection tests were successful with a reduction in detection rates of greater than 6.

Comparing the results from all the tests performed, it can be observed that the encryptors were more successful than the packers (previously tested in Table 5.24). This success is largely owing to PEScrambler which also has a high detection rate, similar to the custom encryptor with the static key which was almost never detected.

In conclusion, this chapter demonstrated the effectiveness of encrypting a malicious binary in a manner so that it can evade an antivirus application. The goal of an average reduction of 10 percentage points was achieved with an average reduction of 29 percentage points. The chapter used the processes and methodologies detailed in Chapter 3 and Chapter 4. Having demonstrated the effectiveness of encryption, the next chapter will attempt to achieve better results by combining the techniques used in this chapter (encryption) and the techniques used in Chapter 5 (packing).

Chapter 7

Evasion : Combination

7.1 Introduction

Chapters 5 and 6 tested the packing and encryption techniques using the test protocol applications as defined in chapter 4. This chapter will investigate if combining a packer and an encryptor on the same binary will reduce the detection rate. The testing will be performed in two sets, evaluating whether packing a binary first or encrypting the binary first produces a binary that has a higher percentage of evasion. These tests are performed since they provide different output binaries depending on which evasion technique is applied first.

7.2 Hypothesis

By combining both the packer and encryption applications on the same binary, the detection rates will be reduced when compared to the original single instance tests. There are two possible outcomes that are expected from this test. The first possible outcome is that there will be a positive reaction to the combination of the packer and encryptor, which will result in a reduced detection rate when compared to the original tests. The second possible outcome is that the antivirus engines detect either the packer or encryptor resulting in an increased detection rate.

7.2.1 Goals

In this chapter it is favorably expected that based on tests in the previous two chapters in which the packers and encrypters were tested separately, the percentage point changes can be found in tables 7.1 and 7.2.

Table 7.1: Expected Goals For Pack First Tests

Pack first	**ASPack**	**UPX**	**PECompact**
Baseline	13pp	11pp	10pp
Baseline OP	-5pp	-14pp	7pp
Malicious Binary 1	25pp	15pp	25pp
Malicious Binary 2	30pp	22pp	31pp
Malicious Binary 3	30pp	0pp	30pp
Average	19pp	7pp	21pp

Table 7.2: Expected Goals For Encrypt First Tests

Encrypt First	**Hyperion**	**PEScrambler**
Baseline	-1pp	9pp
Baseline OP	0pp	9pp
Malicious Binary 1	0pp	35pp
Malicious Binary 2	22pp	25pp
Malicious Binary 3	0pp	39pp
Average	4.2pp	23.4pp

Tests that have a 0 percentage point change recorded in tables 7.1 and 7.2, indicate binaries that were not able to encrypted or packed previously and are unlikely to have a different result when combined with the alternate packer or encryptor. The remainder of the binaries were give a percentage point goal of half the of the original tests results, this is due to the possibility that the combination of a packer or encryptor is likely to affect the original detection score. While its not possible to say exactly how it will affect the original score, it is expected that in most cases it will cause the binary to be flagged more often. In cases where it is not flagged it is then expected that the score will demonstrate better evasion scores.

7.3 Tests

The tests will be performed in two sets. First the binary will be packed and then encrypted for each of the existing packers and encryptors. This will be followed by a set of tests for the opposite order, the binary will be encrypted and then packed. By performing both sets of tests, it can be verified as to what exactly the antivirus engines are checking for. If the tests results in both tests are exactly the same, then it will be assumed that

the antivirus engines are actually extracting the packed data and allowing the encrypted data to hit this disk. If the tests result in differing data, then it can be assumed that the antivirus applications are only verifying one level of maliciousness and reporting this first level of scanning only. The testing in the upcoming sections will test the binaries listed in the Table 4.14. It should be noted that the tests in this chapter will only be performed when the binaries that are being tested can be packed and encrypted. If either one is not possible, then the binary will be excluded from that test as it will have the same results as the original test performed in either the Chapter 5 on packing or Chapter 6 on encrypting.

7.4 Pack-First Tests

This section will be broken into the subsections listed in Table 7.3 which deals with a cross paired test between all the packers and encryptors previously covered in Chapters 5 and 6.

Table 7.3: Listing of Pack First Tests

Sub Section	**Binary First Packed With**	**Result Is Then Encrypted With**
7.4.1	UPX	Hyperion
7.4.2	UPX	PEScrambler
7.4.3	ASPack	Hyperion
7.4.4	ASPack	PEScrambler
7.4.5	PECompact	Hyperion
7.4.6	PECompact	PEScrambler

7.4.1 UPX - Hyperion

The results from scanning the binaries with the combination of UPX and Hyperion are found in Table 12. The base template had a marginal increase of 7 percentage points, this is probably the same as the Hyperion increase. The opcode template had a slight increase in detection of 2 percentage points. The first malicious binary was incapable of being packed using UPX and as such was not considered for testing. Malicious Binary number 2 had a reasonable reduction in detection rates of 55 percentage points which is on par with the original UPX reduction results. The third malicious binary had the same results as the first binary in terms of reduction of detection rates, with a reduction of 53 percentage points.

7.4.2 UPX - PEScrambler

The results from scanning the binaries with the combination of UPX and PEScrambler are found in Table 13. When scanning the baseline binary with a combination of UPX and PEScrambler, a reduction in detection rates of 22 percentage points was observed when compared to the original scan results of the baseline binary. When scanning template base with the combination of UPX and PEScrambler, there is an increase of 25 percentage points in the detection rate which contradicts what was expected after observing the previous scan results. When scanning Malicious Binary number 1 with the combination of UPX and PEScrambler, it showed a decrease of 35 percentage points in its detection rate and was close to the original PEScrambler detection results. When scanning the combination of UPX and PEScrambler, Malicious Binary number 2 showed a decrease of 44 percentage points in its detection rate was close to the original PEScrambler detection results. When scanning the combination of UPX and PEScrambler, Malicious Binary number 3 showed a decrease of 78 percentage points in its detection rate and was close to the original PEScrambler results.

7.4.3 ASPack - Hyperion

The results from scanning the binaries with the combination of ASPack and Hyperion are found in Table 14. The base Metasploit template scan shows an increase of 4 percentage points in its detection rates when manipulated by ASPack and Hyperion. This is the same effect as observed for UPX and Hyperion, with the exception that the increase is not as large. The Metasploit opcode scan shows a tiny increase in detection rates as with the previous test with an increase of 2 percentage points. It is interesting to note that the increased percentage is exactly the same as the UPX and Hyperion test. As with previous tests, there was a significant decrease in the detection rates of the malicious binaries. The first of the malicious binaries was not able to be packed and as such had no results. However a decrease of 55 percentage points for Malicious Binary number 2. As with previous tests, there was a significant decrease in the detection rates of the malicious binaries, with a decrease of 47 percentage points in detection rates for Malicious Binary number 3.

7.4.4 ASPack - PEScrambler

The results from scanning the binaries with the combination of ASPack and PEScrambler are found in Table 15. When performing the combination test with ASPack and PEScrambler on the base template, a significant reduction in detection rates of 27 percentage points was observed. For the first time this result shows a lower detection rate than that of

the original PEScrambler or ASPack test results. When performing the test with the Metasploit opcode binary, a significant reduction in detection rates of 18 percentage points was observed. When performing the test with the first malicious binary an average reduction of 49 percentage points in detection rates is observed. When performing the test with the second malicious binary, a significant reduction of 62 percentage points in detection rates is observed. The final test performed with Malicious Binary number 3 also results in a significant reduction of 82 percentage points in detection rates. This is also the first time that one of the malicious binaries has shown such a high reduction in its rate of detection. It is greater than that observed in the original PEScrambler or ASPack results.

7.4.5 PECompact - Hyperion

The results from scanning the binaries with the combination of PECompact and Hyperion are found in Table 16. The Baseline Metasploit test with the current combination resulted in an increased detection rate of 7 percentage points in the rate of detection. When performed with the Baseline Metasploit opcode binary using the current combination, an increase of 5 percentage points in the rate of detection was observed. When testing the malicious binaries, only malicious binary number two and malicious binary number three successfully packed and encrypted with the current combination being tested. These binaries showed a decrease of 49 percentage points in their detection rates.

7.4.6 PECompact - PEScrambler

The results from scanning the binaries with the combination of PECompact and PEScrambler are found in Table 17. The basic template was not able to successfully have the packer and encryptor evasion applied. When scanned, the Metasploit opcode template showed a reduction of 29 percentage points in the rate of detections. All three binaries successfully had the current combination applied to them and showed a reduction of 82, 69 and 80 percentage points in their detection rates of each of the binaries respectively.

Table 7.4: Summary Table For Pack First

	ASPack		**PECompact**		**UPX**	
Binary	**Hyperion**	**PEScrambler**	**Hyperion**	**PEScrambler**	**Hyperion**	**PEScrambler**
Baseline	24/55 (-4 pp)	7/55 (27 pp)	26/55 (-7 pp)	0/55 (0 pp)	26/55 (-7 pp)	10/55 (22 pp)
Baseline OP	24/55 (-2 pp)	13/55 (18 pp)	26/55 (-5 pp)	7/55 (29 pp)	24/55 (-2 pp)	37/55 (-25 pp)
Malicious Binary 1	0/55 (0 pp)	21/55 (49 pp)	0/55 (0 pp)	3/55 (82 pp)	0/55 (0 pp)	29/55 (35 pp)
Malicious Binary 2	22/55 (55 pp)	18/55 (62 pp)	25/55 (49 pp)	14/55 (69 pp)	22/55 (55 pp)	28/55 (44 pp)
Malicious Binary 3	26/55 (47 pp)	7/55 (82 pp)	25/55 (49 pp)	8/55 (80 pp)	23/55 (53 pp)	9/55 (78 pp)
Percentage Point Average	19 pp	48 pp	17 pp	52 pp	20 pp	31 pp

7.5 Encrypt-First Tests

This section will test the use of a combination of packer and encryptors as in the previous section 7.4. The difference in the tests can be found in the order in which the packers and encryptors are applied. The previous chapter applied the packers first while this chapter applies the encryptors first. This section will be broken into the subsections listed in Table 7.5 which covers a cross paired test between all the encryptors and packers previously covered in Chapters 5 and 6.

Table 7.5: Listing of Pack First Tests

Sub Section	**Binary First Encrypted With**	**Result Is Then Packed With**
7.5.1	Hyperion	ASPack
7.5.2	Hyperion	PECompact
7.5.3	Hyperion	UPX
7.5.4	PEScrambler	ASPack
7.5.5	PEScrambler	PECompact
7.5.6	PEScrambler	UPX

7.5.1 Hyperion - ASPack

The results from scanning the binaries with the combination of Hyperion and ASPack are found in Table 18. When the base Metasploit template is scanned, a decrease of 11 percentage points in its detection rates was observed. While this is the first test for the "encrypt first" tests, when compared to "pack first" tests it is a promising result as it shows a decrease instead of an increase which was the observed result for many of the "pack first" scans. The next test which is the Metasploit opcode scan. It demonstrated a decrease of 15 percentage points in its detection rates. While this is only marginally better than the base template scan, it shows better results than the alternate paired scan where ASPack and Hyperion were employed. From the three malicious binaries only two were able to have the current combination applied to them. These binaries (Binary number 2 and Binary number 3) showed a reduction of 71 and 73 percentage points respectively in their detection rates.

7.5.2 Hyperion - PECompact

The results from scanning the binaries with the combination of Hyperion and PECompact are found in Table 19. The base template and the opcode code template both successfully had the current combination applied to them and showed a reduction of 20 and 13

percentage points respectively in their detection rates. As with the previous test in subsection 7.5.1, only two of the malicious binaries had the current combination successfully applied to them. These binaries (malicious binary number 2 and malicious binary number 3) both showed a significant reduction of 78 and 80 percentage points in their detection rates.

7.5.3 Hyperion - UPX

The results from scanning the binaries with the combination of Hyperion and UPX are found in Table 20. The base template and the opcode code template both successfully had the current combination applied to them and showed a reduction of 5 and 7 percentage points respectively in their detection rates. This reduction is the lowest reduction for the base tests with any of the current "encrypt first" combinations. Of the three malicious binaries, only the third malicious binary successfully had the current combination applied to it. This combination did result in a significant detection rate reduction of 69 percentage points.

7.5.4 PEScrambler - ASPack

The results from scanning the binaries with the combination of PEScrambler and ASPack are found in Table 21. The base template and the opcode code template both successfully had the current combination applied to them and showed a reduction in detection rates of 24 and 29 percentage points respectively in their detection rates. All three of the malicious binaries were able to have the current combination of evasions applied to it. All three of the binaries had a significant reduction in detections rates, these were 65, 76 and 95 percentage points respectively in their detection rates. This is the only time a scan has ever resulted in zero detections. It should be noted that this combination which results in a 100% reduction only seems to work for Malicious Binary number 3, as the other binaries that were tested with the same combination did not result in a 100% reduction

7.5.5 PEScrambler - PECompact

The results from scanning the binaries with the combination of PEScrambler and PECompact are found in Table 22. The base template and the opcode code template both successfully had the current combination applied to them and showed a reduction in detection rates of 20 and 27 percentage points respectively. Of the three malicious binaries, only Malicious Binary numbers 1 and 2 successfully had the current evasions applied. These binaries showed a reduction in their detection rates of 78 and 69 percentage points respectively.

7.5.6 PEScrambler - UPX

The results from scanning the binaries with the combination of PEScrambler and UPX are found in Table 23. The base template and the opcode code template both successfully had the current combination applied to them and showed a reduction in its detection rates of 13 and 24 percentage points respectively. Of the three malicious binaries, only Malicious Binary 1 and 2 successfully had the current evasions applied. These binaries showed a reduction in their detection rates of 67 and 56 percentage points respectively.

Table 7.6: Summary Table For Encrypt First

	Hyperion			**PEScrambler**		
Binary	ASPack	PECompact	UPX	ASPack	PECompact	UPX
Baseline	16/55 (11 pp)	11/55 (20 pp)	19/55 (5 pp)	9/55 (24 pp)	11/55 (20 pp)	15/55 (13 pp)
Baseline OP	15/55 (15 pp)	16/55 (13 pp)	19/55 (7 pp)	7/55 (29 pp)	8/55 (27 pp)	10/55 (24 pp)
Malicious Binary 1	-/55 (0 pp)	-/55 (0 pp)	-/55 (0 pp)	12/55 (65 pp)	5/55 (78 pp)	11/55 (67 pp)
Malicious Binary 2	13/55 (71 pp)	9/55 (78 pp)	-/55 (0 pp)	10/55 (76 pp)	14/55 (69 pp)	21/55 (56 pp)
Malicious Binary 3	12/55 (73 pp)	8/55 (80 pp)	14/55 (69 pp)	0/55 (95 pp)	-/55 (0 pp)	-/55 (0 pp)
Percentage Point Average	34 pp	38 pp	16 pp	58 pp	39 pp	32 pp

7.6 Summary

After performing all the tests in this chapter the following summary (Table 7.4) can be drawn to show the effectiveness of each combination across the binaries that were tested.

In the “pack first” tests the binaries were first packed and then encrypted with either Hyperion or PEScrambler alternatively. The binaries that could not be packed and encrypted are represented with an N/A in the respective column. The testing showed that in the end only four combinations of a packer and encryptor did not succeed. Hyperion showed an averseness to working with packed binaries as it could not modify the flow correctly and as such would simply fail. This failure to encrypt was only apparent when testing with the first malicious binary.

When analyzing the results for the ASpack and Hyperion combination, it can be stated that it performed reasonably well when compared to the average for the expected goals defined in the beginning of this chapter. The results from tests showed an average evasion of only 24 percentage points which is greater than the expected average of 19 percentage points. The subsequent ASPack test had much better results with an average of 48 percentage across the five binaries. This result exceeds expectations and meets the goal of achieving greater than the 19 percentage points evasion from the original goals.

When analyzing the results for PECompact, the reduction in detection rates is only just greater than the goal (of 21 percentage points) set out in the starting of the chapter

for the PECompact packer with an average of 21.5 percentage points across the four binaries tested. The subsequent test with PEScrambler achieves a much better average detection rate of 65 percentage points. While only nominally better than the results of the first packer, it did achieve the best reduction thus far.

The final packer to be tested in the pack's first section was UPX, which achieved an average set in the middle of the group's detection rate with an average reduction of 28 percentage points. It performed only a little worse than the results achieved by ASPack but performed much better than those achieved by PECompact. Both the combinations for UPX exceeded the goals that were original defined in the start of the chapter of a 7 percentage point decrease.

After performing the tests in the "pack's first" section, the next set of tests focused on first encrypting the binary and then packing the encrypted binary. This method of testing resulted in a number of binaries that could not be packed. This is mostly the case when looking at the first malicious binary, but it also occurs in the other malicious binaries, although less frequently. The outcome of these tests were also the first time that a 100% evasion was achieved.

When performing the test with Hyperion as the first part of the combination, the following results were observed. When combined with ASPack the reduction achieved was higher than the results achieved in the "pack first" tests with an average of 42 percentage points which is greater than the goal that was calculated at the start of the chapter of 4 percentage points. This is an interesting observation as it showed that simply changing the order in which the evasions were applied resulted in a reduced detection rate. The combination with PECompact showed an average reduction of 48 percentage points, which is above the goal that we originally set in the start of the chapter of 4 percentage points. The UPX combination was the lowest performing of all the packers with an average reduction of 27 percentage points but still exceeded the original goal of 4 percentage points.

When testing the PEScrambler combinations, the following results were observed. The ASPack combination resulted in an average reduction of 58 percentage points across the five binaries which exceeds the goal of 23 percentage points that was defined in the start of the chapter. This is the highest reduction when comparing all five binary tests. The PECompact test resulted in a reduction of 49 percentage points in its detection results when compared against four binaries which exceeded the goal of 23 percentage point reduction in the number of reduction. The final test with UPX showed the lowest average reduction of 40 percentage points but still exceeded the goal of a 23 percentage point reduction. As expected UPX was the lowest rated combination.

In conclusion, this chapter demonstrated that it was possible to further reduce the

detection rate by combining the two existing evasion techniques. The next chapter will conclude with what was observed in the results from this chapter as well as previous chapters and present a proposal for future work that can be targeted. We will also reiterate the limitations of the current tests and suggest how they can be further improved.

Chapter 8

Conclusion

8.1 Introduction

The previous chapter showed that the combination testing approach was valuable and demonstrated that the combination of packers and encryptors provided better results than a single evasion technique. This chapter will present a summary of the data from Chapters 5, 6 and 7. The original goals that were set out will then be restated along with the accompanying results. The chapter will be concluded with a summary and a proposal for future research of this subject matter.

8.2 Chapter Summaries

Chapter 2: In this chapter we introduced a number of key terms that are used in the rest of the report. Having introduced these terms, the chapter proceeded to provide a taxonomy on the different techniques used to evade antivirus applications. Following this explanation, the chapter proceeded to demonstrate how the antivirus industry responded to the different evasion techniques. The chapter concluded with the related work in the areas of online scanning and antivirus evaluation research.

Chapter 3: This chapter introduced the options that were available for building the antivirus testbed in the forms of online and offline testing. The chapter then explored the benefits and drawbacks of using a custom offline laboratory and indicated that due to the high demand in resources, it would no longer be pursued as a viable option. This was followed up by the benefits and drawbacks of using an online system in the form of the VirusTotal web application. The chapter concluded by indicating that the online platform would be best suited for testing in the chapters that followed.

Chapter 4: This chapter introduced the process for testing the malicious binary evasion techniques. The chapter outlined the process for testing the evasion techniques by selecting a number of binaries which have a known state before scanning and then monitoring for an expected state after scanning the binary. The chapter then introduced and explained why each of the binaries was selected and what their expected outcomes were. The chapter concluded by selecting the binaries that would be employed in Chapters 5 - 7.

Chapter 5: Chapter 5 described the process of testing the packing evasion techniques and the resulting outcomes. The results illustrated that packing the malicious binaries resulted in a significant reduction in detection rates. The results also showed that none of the packer techniques were able to achieve a 100% evasion against all antivirus applications.

Chapter 6: This chapter continued the testing of the second of two evasion techniques as selected in Chapter 2. As with that chapter, the testing showed that by applying an encryption evasion to the malicious binaries it resulted in a significant reduction in detection rates. Testing with the custom binaries resulted in an increase of detection rates. As with the previous chapter, none of the wrappers that were applied resulted in a 100% reduction in detection.

Chapter 7: The seventh chapter performed a number of tests using a combination of the evasion techniques mentioned in Chapters 5 and 6. These tests were performed to determine if stacking the evasion techniques would result in a decreased detection rate. The results from these tests indicated that when encrypted first, the binaries were more likely to evade an antivirus engine. This was the first chapter in which a 100% evasion was observed when running the tests with the PEScrambler and ASPack combination.

8.3 Research Goals

The original research question in chapter 1 set out to investigate whether binaries which exhibited known virus-like evasion techniques could prove effective against modern antivirus engines when using on-demand scanning techniques. The results from the tests in chapters 5 - 7 demonstrate that the evasion techniques that were tested were not overwhelmingly effective in reducing the detection rates across the four test groups. The tests in chapters 5 - 7 demonstrated that while it is possible to gain high levels of antivirus evasion, these high levels of evasion are very selective. The levels of evasion depend on a number of factors present in the original binary that was submitted for testing. The factors in the

binaries that contributed to a more consistent reduction were larger, more complex and had a high baseline detection rate. This is in comparison to the smaller and simpler control binaries which had a lower starting detection rate and was not able to achieve a consistent reduction in detection rates.

From the tests in chapter 7 it was possible to determine that a number of antivirus engines do react to simple signatures in a binary by simply alternating the evasion technique that was first applied and then evaluating the results. In cases where it was possible to apply both evasion techniques to a binary, higher evasion rates were observed.

The intended goal for Chapters 5, 6 and 7 was to gain increasing evasion rates in each chapter, which would match the complexity of the evasion techniques applied to the test protocols. This began In Chapter 5, where a goal of 10 percentage point reduction was set to be achieved. After reviewing the results, it is observed that this goal was achieved with an average reduction of 36 percentage points. The intended goal for Chapter 6 was to achieve an evasion rate of 10 percentage points or greater, which is the same as the goals for the packers, this was achieved with a reduction of 29 percentage points across the binaries tested.

The goals for Chapter 7 were two-fold. The first goal evaluated the effectiveness of the evasion achieved by the "packer first" combination to determine if a higher evasion rate could be achieved compared to the original tests. Following the same method of testing, the second set of tests evaluated the effectiveness of the evasion achieved by the "encrypt first" combination. The first goal was achieved: the average reduction in evasion rates was 37 percentage points which is greater than the original results of 36 percentage points. The second goal was successfully achieved as well: it averaged a 40 percentage point reduction in detection rates which is greater than the original average reduction of 29 percentage points. It stands that the goals set out were achieved for this chapter. The first section barely achieved the reduction goal with only 1 percentage point greater than the goal, while the second section managed to achieve a better result with an average reduction of 11 percentage points.

8.4 Future Work

There are currently a number of areas that can be expounded upon to take this work forward. The first of these areas is to perform further analysis on the exact costing in terms of time and resources when using an offline laboratory. Once this is completed, the process of automating the laboratory setup with the use of applications such as Docker or Vagrant should be explored. If implemented correctly, this can reduce the testing time for future research completed in this area. Further investigation also needs to be completed in

the areas of the more advanced evasion techniques. This will most likely require acquiring the source code to an existing trojan or virus and modifying it in such a way as to follow the idea behind one of the dynamic evasion methods such as polymorphism.

8.5 Conclusion

In conclusion, the work performed here explored the area of testing whether an evasion technique could bypass an antivirus engine. This work further developed the approach of online testing in which VirusTotal was evaluated as an online testing platform. This testing demonstrated that it is possible to gain a reduction in the detection rates of antivirus engines. It is noted that it was not possible to gain a perfect 100% evasion for all binaries with any of the techniques tested. The work also illustrated that packers have a good chance at producing a binary that can evade antivirus when compared to the same ability in encryptors.

References

Adrion, W. R., Branstad, M. A., and Cherniavsky, J. C. Validation, verification, and testing of computer software. *ACM Computing Surveys*, 14(2):159–192, June 1982. ISSN 0360-0300. doi:10.1145/356876.356879.

Akritidis, P., Markatos, E. P., Polychronakis, M., and Anagnostakis, K. Stride: polymorphic sled detection through instruction sequence analysis. In *Security and Privacy in the Age of Ubiquitous Computing*, pages 375–391. Springer, 2005.

Alme, C. and Eardly, D. Mcafee anti-malware engines: values and technologies. Technical report, McAfee Labs, 2010. Last Accessed: 21 Jan 2015.
URL `http://bit.ly/1CElV12`

Alsagoff, S. Malware self protection mechanism. In *International Symposium on Information Technology*, volume 3, pages 1–8. Aug 2008. doi:10.1109/ITSIM.2008.4631981.

Anderson, J. P. Computer security technology planning study. Technical report, DTIC Document, 1972. Last Accessed: 21 Jan 2015.
URL `http://csrc.nist.gov/publications/history/ande72.pdf`

Angel, D. 40hex e-zine. 1993. Last Accessed: 06 Apr 2013.
URL `http://www.textfiles.com/magazines/40HEX/40hex011`

Aycock, J. Computer viruses and malware. Advances in Information Security. Springer, 2006. ISBN 9780387341880.

Aycock, J., Degraaf, R., and Jacobson, M. Anti-disassembly using cryptographic hash functions. In *Journal in Computer Virology*, volume 2, pages 1–8. Springer, 2006.

Beroset, E. The nuke encryption device. In *Virus Bulletin Conference*. November 1993.

Bishop, P., Bloomfield, R., Gashi, I., and Stankovic, V. Diversity for security: a study with off-the-shelf antivirus engines. In *International Symposium on Software Reliability Engineering*. Nov 2011. ISSN 1071-9458. doi:10.1109/ISSRE.2011.15.

Borello, J.-M., Filiol, E., and Me, L. From the design of a generic metamorphic engine to a black-box classification of antivirus detection techniques. *Journal in Computer Virology*, 6(3):277–287, 2010. ISSN 1772-9890. doi:10.1007/s11416-009-0136-2.

Branco, R. R., Barbosa, G. N., and Neto, P. D. Scientific but not academical overview of malware. In *Black Hat Technical Security Conference*. 2012. Last Accessed: 11 July 2015.
URL `https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_Slides.pdf`

Brand, M. Analysis avoidance techniques of malicious software. Ph.D. thesis, Edith Cowan University, 2010.

Brulez, N. Win32 portable executable packing uncovered. Technical report, Websense Security Labs, 2009. Last Accessed: 21 Jan 2015.
URL `http://bit.ly/15v7O3t`

Burgess, J. The tradition of the trojan war in homer and the epic cycle. The Tradition of the Trojan War in Homer and the Epic Cycle. Johns Hopkins University Press, 2003. ISBN 9780801874819.

Bustamante, P., Urzay, I., Corrons, L., and Franco, J. From traditional antivirus to collective intelligence. Technical report, Panda Security, 2007. Last Accessed: 01 Dec 2014.
URL `http://bit.ly/15v7Okf`

Chien, E. and Ször, P. Blended attacks exploits, vulnerabilities and buffer-overflow techniques in computer viruses. In *Virus Bulletin Conference*, pages 1–36. 2002.

Chiriac, M. Tales from cloud nine. In *Virus Bulletin Conference*, pages 1–6. 2009.

Christodorescu, M. and Jha, S. Testing malware detectors. In *ACM Special Interest Group on Software Engineering Notes*, volume 29, pages 34–44. ACM, 2004.

Cohen, F. Computer viruses. *Computers & Security*, 6(1):22–35, February 1987. ISSN 0167-4048. doi:10.1016/0167-4048(87)90122-2.

Davis, T. Utilizing entropy to identify undetected malware. Technical report, Cybersecurity Solutions, 2009. Last Accessed: 21 Jan 2015.
URL `http://bit.ly/1y0TnuA`

Dinaburg, A., Royal, P., Sharif, M., and Lee, W. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62. ACM, 2008.

Doherty, S. and Gegeny, J. Hidden lynx professional hackers for hire. Technical report, Symantec, 2013. Last Accessed: 21 Jan 2015.
URL `http://wrd.cm/1CSpLnF`

Ferrie, P. and Ször, P. Zmist opportunities. In *Virus Bulletin*, volume 3, pages 6–7. 2001.

Filiol, E. Strong cryptography armoured computer viruses forbidding code analysis: the bradley virus. Research Report RR-5250, Institut National De Recherche En Informatique Et En Automatique, 2004. Last Accessed: 01 Dec 2014.
URL `https://hal.inria.fr/inria-00070748`

Fuhs, H. Encryption generators used in computer viruses. 1995. Last Accessed: 01 Dec 2014.
URL `http://www.fuhs.de/en/pub/encryptgen1.shtml`

Funk, C. and Garnaeva, M. Kaspersky security bulletin 2013. overall statistics for 2013. *Securelist*, 2013. Last Accessed: 11 July 2015.
URL `https://securelist.com/analysis/kaspersky-security-bulletin/58265/kaspersky-security-bulletin-2013-overall-statistics-for-2013/`

Garnaeva, M., Chebyshev, V., Makrushin, D., Unuchek, R., and Ivanov, A. Kaspersky security bulletin 2014. *Securelist*, 2014.

Gashi, I., Stankovic, V., Leita, C., and Thonnard, O. An experimental study of diversity with off-the-shelf antivirus engines. In *Eighth IEEE International Symposium on Network Computing and Applications*, pages 4–11. IEEE, 2009.

Global Research Analysis Team, G. The virus top 20 for november 2001. 2001. Last Accessed: 20 Jan 2015.
URL `http://bit.ly/1ta726V`

Goodin, D. Puzzle box part 1. 2014a. Last Accessed: 01 Dec 2014.
URL `http://bit.ly/1BuAWWm`

Goodin, D. Puzzle box part 2. 2014b. Last Accessed: 01 Dec 2014.
URL `http://bit.ly/1xMGUL1`

Gordon, S. and Chess, D. Where there's smoke, there's mirrors: the truth about trojan horses on the internet. In *Virus Bulletin International Conference Proceedings*. 1998.

Gordon, S. and Chess, D. Attitude adjustment: trojans and malware on the internet. In *Proceedings of the European Institute for Computer Antivirus Research*, pages 183–204. 1999.

Gordon, S. and Ford, R. Real world anti-virus product reviews and evaluations–the current state of affairs. In *Proceedings of the 1996 National Information Systems Security Conference*. 1996.

Gryaznov, D. Scanners of the year 2000: heuristics. In *Proceedings of the 5th International Virus Bulletin*, volume 113. 1999. Last Accessed: 01 Dec 2014.
URL `http://ivanlef0u.fr/repo/madchat/vxdevl/vdat/epscan20.htm`

Guo, F., Ferrie, P., and Chiueh, T.-C. A study of the packer problem and its solutions. In *Recent Advances in Intrusion Detection*, pages 98–115. Springer, 2008.

Haffejee, J. and Irwin, B. Testing antivirus engines to determine their effectiveness as a security layer. In *Information Security for South Africa (ISSA), 2014*, pages 1–6. Aug 2014. doi:10.1109/ISSA.2014.6950496.

Harper, A., Harris, S., Ness, J., Eagle, C., Lenkey, G., and Williams, T. Gray hat hacking the ethical hackers handbook, 3rd edition. McGraw-Hill Education, 2011. ISBN 9780071742566.

Herm1t. Virus ezine listing. 2002. Last Accessed: 01 Dec 2014.
URL `http://vxheaven.org/lib/static/vdat/ezinemen.htm`

Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H., and Chen, C.-W. Antivirus software shield against antivirus terminators. *Information Forensics and Security, IEEE Transactions on*, 7(5):1439–1447, Oct 2012. ISSN 1556-6013. doi:10.1109/TIFS.2012.2206028.

Hypponen, M. Overview of the brain virus. 2011. Last Accessed: 01 Dec 2014.
URL `http://campaigns.f-secure.com/brain/virus.html`

Hypponen, M. Overview of the cascade virus. 2014. Last Accessed: 01 Dec 2014.
URL `http://www.f-secure.com/v-descs/cascade.shtml`

Kang, M. G., Poosankam, P., and Yin, H. Renovo: a hidden code extractor for packed executables. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode*,

WORM '07, pages 46–53. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-886-2. doi:10.1145/1314389.1314399.

Kephart, J. and White, S. Directed-graph epidemiological models of computer viruses. In *Computer Society Symposium on Research in Security and Privacy*, pages 343–359. May 1991. doi:10.1109/RISP.1991.130801.

Kephart, J. O. and Arnold, W. C. Automatic extraction of computer virus signatures. In *Virus bulletin international conference*, pages 178–184. 1994.

Kienzle, D. M. and Elder, M. C. Recent worms: a survey and trends. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, WORM '03, pages 1–10. ACM, New York, NY, USA, 2003. ISBN 1-58113-785-0. doi:10.1145/948187.948189.

Knight, C. What is an e-zine. 2005. Last Accessed: 01 Dec 2014.
URL `http://emailuniverse.com/ezine-tips/?id=1312`

Konstantinou, E. and Wolthusen, S. Metamorphic virus: analysis and detection. *Royal Holloway University of London*, 15, 2008.

Kramer, S. and Bradfield, J. C. A general definition of malware. *Journal in Computer Virology*, 6(2):105–114, 2010. ISSN 1772-9890. doi:10.1007/s11416-009-0137-1.

Krebs, B. Virus scanners for virus authors. 2009. Last Accessed: 01 Dec 2014.
URL `http://krebsonsecurity.com/2009/12/virus-scanners-for-virus-authors/`

Lammer, P. 1260 revisited. In *Virus Bulletin Conference*, page 12. March 1990.

Leder, F., Steinbock, B., and Martini, P. Classification and detection of metamorphic malware using value set analysis. In *Malicious and Unwanted Software, 2009 4th International Conference on*, pages 39–46. Oct 2009. doi:10.1109/MALWARE.2009.5403019.

Lin, J. On malicious software classification. In *Intelligent Information Technology Application Workshops*, pages 368–371. Dec 2008. doi:10.1109/IITA.Workshops.2008.106.

Little, M. Tealab: a testbed for ad hoc networking security research. In *Military Communications Conference*, pages 936–942 Vol. 2. Oct 2005. doi:10.1109/MILCOM.2005.1605800.

Lyda, R. and Hamrock, J. Using entropy analysis to find encrypted and packed malware. *IEEE Security Privacy*, 5(2):40–45, March 2007. ISSN 1540-7993. doi:10.1109/MSP.2007.48.

McAfee. Potentially unwanted programs. 2005. Last Accessed: 01 Dec 2014. URL `http://bit.ly/1uwAk0A`

McAfee. Malware report for the whale virus. 2014a. Last Accessed: 01 Dec 2014. URL `http://bit.ly/1uwA6Xk`

McAfee. Malware report for v2px. 2014b. Last Accessed: 01 Dec 2014. URL `http://bit.ly/1yPKLxP`

Moore, T., Clayton, R., and Anderson, R. The economics of online crime. *The Journal of Economic Perspectives*, 23(3):3–20, 2009.

Nachenberg, C. Understanding and managing polymorphic viruses. *The Symantec Enterprise Papers*, 30:16, 1996.

Nachenberg, C. Computer virus-antivirus coevolution. *Communications of the ACM*, 40(1):46–51, January 1997. ISSN 0001-0782. doi:10.1145/242857.242869.

Oberheide, J., Bailey, M., and Jahanian, F. Polypack: an automated online packing service for optimal antivirus evasion. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies*, WOOT'09, pages 9–9. USENIX Association, Berkeley, CA, USA, 2009.

Oberheide, J., Cooke, E., and Jahanian, F. Rethinking antivirus: executable analysis in the network cloud. In *Proceedings of the 2Nd USENIX Workshop on Hot Topics in Security*, HOTSEC'07, pages 5:1–5:5. USENIX Association, Berkeley, CA, USA, 2007.

Oberheide, J., Cooke, E., and Jahanian, F. Cloudav: n-version antivirus in the network cloud. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 91–106. USENIX Association, Berkeley, CA, USA, 2008.

OECD. Computer viruses and other malicious software a threat to the internet economy: a threat to the internet economy. OECD Publishing, 2009. ISBN 9789264056503.

O'Kane, P., Sezer, S., and McLaughlin, K. Obfuscation: the hidden malware. *IEEE Security Privacy*, 9(5):41–47, Sept 2011. ISSN 1540-7993. doi:10.1109/MSP.2011.98.

Parikka, J. Digital Contagions: A Media Archaeology of Computer Viruses. Digital formations. Peter Lang, 2007. ISBN 9780820488370.

Pearce, S. Intro to polymorphisim. 2003. Last Accessed: 01 Dec 2014. URL `http://bit.ly/1JbrHvl`

Perdisci, R., Lanzi, A., and Lee, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941–1946, 2008.

Petik. Interview with the mental driller from 29a. 2002. Last Accessed: 01 Dec 2014.

Quist, D. and Smith, V. Covert debugging circumventing software armoring techniques. In *Black hat briefings USA*. 2007.

Rad, B. B., Masrom, M., and Ibrahim, S. Evolution of computer virus concealment and anti-virus techniques: short survey. *Computing Research Repository*, abs/1104.1070, 2011.

Rad, B. B., Masrom, M., and Ibrahim, S. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security*, 12(8):74–83, 2012.

Radhakrishnan, D. Approximate disassembly. Master's thesis, San Jose State University, 2010.

Ramilli, M. and Bishop, M. Multi-stage delivery of malware. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 91–97. IEEE, 2010.

Rescorla, E. Security holes... who cares? In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, (SSYM'03), pages 6–6. USENIX Association, Berkeley, CA, USA, 2003.

Rin, N. Virtual machines detection enhanced. 2013. Last Accessed: 01 Dec 2014. URL `http://artemonsecurity.com/vmde.pdf`

Roundy, K. A. and Miller, B. P. Binary-code obfuscations in prevalent packer tools. *ACM Computing Surveys*, 46(1):4:1–4:32, July 2013. ISSN 0360-0300. doi:10.1145/2522968.2522972.

Royal, P., Halpin, M., Dagon, D., Edmonds, R., and Lee, W. Polyunpack: automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22Nd Annual Computer Security Applications Conference*, ACSAC '06, pages 289–300. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2716-7. doi:10.1109/ACSAC.2006.38.

Saleh, M., Ratazzi, E., and Xu, S. Instructions-based detection of sophisticated obfuscation and packing. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 1–6. Oct 2014. doi:10.1109/MILCOM.2014.9.

Sanok, D. J., Jr. An analysis of how antivirus methodologies are utilized in protecting computers from malicious code. In *Proceedings of the 2nd Annual Conference on Information Security Curriculum Development*, InfoSecCD '05, pages 142–144. ACM, New York, NY, USA, 2005. ISBN 1-59593-261-5. doi:10.1145/1107622.1107655.

Schiffman, M. A brief history of malware obfuscation. 2010. Last Accessed: 01 Dec 2014.
URL `http://bit.ly/1CSq7uF`

Shafiq, M. Z., Tabish, S., and Farooq, M. Pe-probe: leveraging packer detection and structural information to detect malicious portable executables. In *Proceedings of the Virus Bulletin Conference*, pages 29–33. 2009.

Sikorski, M. and Honig, A. Practical malware analysis: the hands-on guide to dissecting malicious software. No Starch Press, 2012. ISBN 9781593274306.

Skulason, F. 1260 - the variable virus. In *Virus Bulletin Conference*, page 12. 1990a.

Skulason, F. Whale a dinosaur heading for extinction. In *Virus Bulletin*, page 17. November 1990b.

Solomon, A. A brief history of pc viruses. *Computer Fraud & Security Bulletin*, 1993(12):9–19, 1993.

Staniford, S., Paxson, V., and Weaver, N. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167. USENIX Association, Berkeley, CA, USA, 2002. ISBN 1-931971-00-5.

Sukwong, O., Kim, H., and Hoe, J. Commercial antivirus software effectiveness: an empirical study. *Computer*, 44(3):63–70, March 2011. ISSN 0018-9162. doi:10.1109/MC.2010.187.

Swart, I. Practical application of open source frameworks to achieve anti-virus avoidance. Academic Conferences International Ltd, 2012.

Ször, P. Nexix_der: Tracing the vixen. In *Virus Bulletin*. April 1996.

Ször, P. The new 32-bit medusa. In *Virus Bulletin*, 2000, pages 8–10. 2000.

Ször, P. The art of computer virus research and defense. Pearson Education, 2005. ISBN 9780672333903.

Ször, P. and Ferrie, P. Hunting for metamorphic. In *Virus Bulletin Conference*. 2001.

Ször, P. and Ferrie, P. Striking similarites: win32/simile and metamorphic virus code. In *Virus Bulletin Conference*. 2002.

Thengade, A., Khaire, A., Mitra, D., and Goyal, A. Virus detection techniques and their limitations. In *International Journal of Scientific & Engineering Research*, volume 5. October 2014.

Thompson, C. The virus underground. *Virus*, 2004. Last Accessed: 10 Oct 2014. URL `https://vxheaven.org/lib/pdf/The%20Virus%20Underground.pdf`

Tutte, W. T. Fish and I. Springer, 2000. Last Accessed: 10 Oct 2014. URL `http://cryptocellar.web.cern.ch/cryptocellar/tutte.pdf`

VirusTotal. Virustotal supported scanners. 2014. Last Accessed: 10 Oct 2014. URL `https://www.virustotal.com/en/about/credits/`

Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 11–18. ACM, 2003.

White, S. R., Kephart, J. O., and Chess, D. M. Computer viruses: A global perspective. In *Proceedings of the Fifth International Virus Bulletin Conference, Boston*, pages 185–191. 1995.

Xu, J., Sung, A. H., Mukkamala, S., and Liu, Q. Obfuscated malicious executable scanner. *Journal of Research and Practice in Information Technology*, 39(3):181–197, 2007.

You, I. and Yim, K. Malware obfuscation techniques: a brief survey. In *BWCCA*, pages 297–300. 2010.

Young, A. and Yung, M. Cryptovirology: extortion-based security threats and countermeasures. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 129–. IEEE Computer Society, Washington, DC, USA, 1996. ISBN 0-8186-7417-2.

Yu, W., Zhang, N., Fu, X., and Zhao, W. Self-disciplinary worms and countermeasures: modeling and analysis. *IEEE Transactions on Parallel and Distributed Systems*, 21(10):1501–1514, October 2010. ISSN 1045-9219. doi:10.1109/TPDS.2009.161.

Zolkipli, M. F. and Jantan, A. Malware behavior analysis: learning and understanding current malware threats. In *Proceedings of the 2010 Second International Conference*

on Network Applications, Protocols and Services, NETAPPS '10, pages 218–221. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-0-7695-4177-8. doi:10.1109/NETAPPS.2010.46.

Glossary

PUP Potentially unwanted application. These applications that are not inherently malicious and their uses depend on the person using the application.. 6

trojan A type of non-viral malicious application that takes its name from Greek mythology. Commonly used to deliver other malicious applications. 6

virus A virus is usually characterised by a small application that is designed to spread by injecting malicious code into other binaries. 7, 97

worm A self-propagating program that attempts to spread by exploiting vulnerabilities in a computer system. 8

Appendices

The following tables refer to the more verbose descriptions and various results of tests that are included in the paper. They are ordered in the order they occur in the paper for easier reference.

Table 1: Metasploit UPX OP Scan Results

Antivirus Engine	Version	Detected As
Antivirus	Version	Malware Signature
MicroWorld-eScan	12.0.250.0	Gen:Variant.Graftor.113722
CAT-QuickHeal	12.00	Trojan.Swrort
McAfee	6.0.4.564	RDN/Generic.dx!c2s
K7Antivirus	9.176.11663	Riskware (0040f0f51)
K7GW	9.176.11663	Riskware (0040f0f51)
TheHacker		Posible_Worm32
NANO-Antivirus	0.28.0.58873	Trojan.Win32.Inject1.cugnjk
TrendMicro-HouseCall	9.700-1001	TROJ_GEN.F0C2C00A314
Kaspersky	12.0.0.1225	HEUR:Trojan.Win32.Generic
BitDefender	7.2	Gen:Variant.Graftor.113722
Ad-Aware	12.0.163.0	Gen:Variant.Graftor.113722
Emsisoft	3.0.0.596	Gen:Variant.Graftor.113722 (B)
Comodo	18058	UnclassifiedMalware
F-Secure	11.0.19100.45	Gen:Variant.Graftor.113722
DrWeb	7.00.8.02260	Trojan.Inject1.33413
VIPRE	28078	Trojan.Win32.Generic!BT
AntiVir	7.11.141.154	TR/Swrort.A.9181
TrendMicro	9.740-1012	TROJ_GEN.F0C2C00A314
McAfee-GW-Edition	2013	RDN/Generic.dx!c2s
Sophos	4.98.0	Mal/Generic-S
Antiy-AVL	0.1.0.1	Trojan[:HEUR]/Win32.AGeneric
Kingsoft	2013.04.09.267	Win32.Troj.Undef.(kcloud)
Microsoft	1.10401	Trojan:Win32/Swrort.A
GData	24	Gen:Variant.Graftor.113722
Panda	10.0.3.5	Generic Malware
Rising	25.0.0.11	PE:HackTool.Swrort!1.6477
Ikarus	T3.1.5.6.0	Trojan.Win32.Swrort
Fortinet	4	Malware_fam.NB
Baidu-International	3.5.1.41473	Trojan.Win32.Swrort.A

Table 2: Malicious Binaries Packed with ASPack

Binary Name	Shortened SHA256 Hash	Detection Ratio
Malicious Binary 1	615863572d46...	16/55
Malicious Binary 2	6e2c9ecad006...	18/55
Malicious Binary 3	7f4cf838c55b...	19/55

Table 3: Malicious Binaries Packed with PECompact

Binary	Shortened SHA256 Hash	Detection Ration
Malicious Binary 1	16a5d346d22c...	18/55
Malicious Binary 2	ad2fcbda013c...	18/55
Malicious Binary 3	d235fb45b4d8...	19/55

Table 4: Malicious Binaries Packed with Custom Packer

Binary	Shortened SHA256 Hash	Detection Ration
Malicious Binary 1	a7db409d5bfe...	13/55
Malicious Binary 2	e12c09d7cc3d...	14/55
Malicious Binary 3	60982b7de65b...	18/55

Table 5: Malicious Binaries Packed with PEScrambler

Binary	Shortened SHA256 Hash	Detection Ratio	Percentage Point Decrease
Malicious Binary 1	4f3aa42ddd71...	8/55	73 pp
Malicious Binary 2	58659313b855...	21/55	56 pp
Malicious Binary 3	b5a91d0508b3...	9/55	78 pp

Table 6: Baseline Scan with UPX

Shortened SHA256 Hash	f7bdb777d1ed...
File name	cmdupx.exe
Detection ratio	2/55

Table 7: Baseline Scan with UPX Details

Antivirus	Result
Bkav	HW32.Paked.B0EF
TheHacker	Posible_Worm32

Table 8: Baseline Scan with ASPack

Shortened SHA256 Hash	d650f9937c35...
File name	cmdasp.exe
Detection ratio	2/55

Table 9: Baseline Scan with UPX Details

Antivirus	Result
CMC	Hoax.Win32.BadJoke.ScreenFlicker!O
Norman	Suspicious.C6!genr

Table 10: Baseline Scan with PECompact

Shortened SHA256 Hash	793d8edfeddb...
File name	cmdpe.exe
Detection ratio	9/55

Table 11: Baseline Scan with UPX Details

Antivirus	Detected As
Agnitum	Packed/PECompact
Bkav	HW32.Paked.ABF3
Cyren	W32/SysVenFak.B.gen!Eldorado
F-Prot	W32/SysVenFak.B.gen!Eldorado
K7AntiVirus	Trojan (00361abb1)
K7GW	Trojan (00361abb1)
McAfee-GW-Edition	BehavesLike.Win32.Rungbu.cc
Norman	Suspicious.C4!genr
Qihoo-360	Malware.QVM17.Gen

Table 12: Results From Dual Scanning Test Binaries with UPX and Hyperion

Shortened SHA256 Hash	Detection Rate	Percentage Point Change
280bdb09196c...	26/55	
c7f050d01718...	24/55	
7c0f676b2100...	22/55	
968e6ae557a6...	23/55	

Table 13: Results From Dual Scanning Test Binaries with UPX and PEScrambler

Shortened SHA256 Hash	Detection Rate
8651c82dd82b...	10/55
d1a4ce55b39b...	37/55
af3cd521b0ba...	29/55
b7d205023174...	28/55
b5a91d0508b3...	9/55

Table 14: Results From Dual Scanning Test Binaries with ASPack and Hyperion

Shortened SHA256 Hash	Detection ratio
9bae621660b9...	24/55
7a5fdbf978c9...	24/55
17cc27e38cd1...	22/55
da4e062d2627...	26/55

Table 15: Results From Dual Scanning Test Binaries with ASPack and PEScrambler

Shortened SHA256 Hash	Detection ratio
7be35e2d8d52...	7/55
59275c39364b...	13/55
5bb572d9a688...	21/55
c97b878c1c57...	18/55
dcd8806f281c...	7/55

Table 16: Results From Dual Scanning Test Binaries with PECompact and Hyperion

Shortened SHA256 Hash	Detection ratio
c46e07ed1798...	26/55
2733768faf9a...	26/55
f7451086ec6a...	25/55
e9b626e7400c...	25/55

Table 17: Results From Dual Scanning Test Binaries with PECompact and PEScrambler

Shortened SHA256 Hash	Detection ratio
fc125bea5928...	7/55
77f260470677...	3/55
2fbe0f4e1e4c...	14/55
87be1e70a35e...	8/55

Table 18: Results From Dual Scanning Test Binaries with Hyperion and PEScrambler

Shortened SHA256 Hash	Detection ratio
9b6f3e81a7e6585c...	16/55
a2e41c4a6b48f6f1...	15/55
67adcf0567b2afef...	13/55
05c531a174064701...	12/55

Table 19: Results From Dual Scanning Test Binaries with Hyperion and PECompact

Shortened SHA256 Hash	Detection ratio
e0b60f4e018a3dfb...	11/55
8f793aff79e0c06d...	16/55
67c2bc5926503461...	9/55
3490bb948eb819d8...	8/55

Table 20: Results From Dual Scanning Test Binaries with Hyperion and UPX

Shortened SHA256 Hash	Detection ratio
136c5d6480b5afaa...	19/55
64fd4ba460d6c199...	19/55
ee83e519a090c6aa...	14/55

Table 21: Results From Dual Scanning Test Binaries with PEScrambler and ASPack

Shortened SHA256 Hash	Detection ratio
ca952fb80b7e7d46...	9/55
68486315fed5b227...	7/55
defb7d5a839dda74...	12/55
d83cd65920b252cb...	10/55
f8d81ee3ef0edc85...	0/55

Table 22: Results From Dual Scanning Test Binaries with PEScrambler and PECompact

Shortened SHA256 Hash	Detection ratio
f8684340d11a...	11/55
60e3f4287ff4...	8/55
c45bdfa113c0...	5/55
38173616b668...	14/55

Table 23: Results From Dual Scanning Test Binaries with PEScrambler and UPX

Shortened SHA256 Hash	Detection ratio
165e7775fd36...	15/55
424cf0a68727...	10/55
0c19dd1e9fd2...	11/55
65e4e92e099d...	21/55

Table 24: Custom Built Template Scan

Shortened SHA256 Hash	8eac4e5b577c...
File name	template_gcc_upx.exe
Detection ratio	10 / 55

Table 25: Basic Metasploit Template Scan

anti-virus	Result	Update
Ad-Aware	Gen:Variant.Graftor.113722	20131208
BitDefender	Gen:Variant.Graftor.113722	20131208
Emsisoft	Gen:Variant.Graftor.113722 (B)	20131208
GData	Gen:Variant.Graftor.113722	20131208
MicroWorld-eScan	Gen:Variant.Graftor.113722	20131208
TheHacker	Posible_Worm32	20131204
TrendMicro	PAK_Generic.001	20131208
TrendMicro-HouseCall	PAK_Generic.001	20131208

Table 26: UPX OP Code Execution Scan Results

Shortened SHA256 Hash	d1a4ce55b39b...
File name	template_gcc_op_upx.exe
Detection ratio	37 / 55