INCREASING CHANCES OF SURVIVAL FOR MALWARE USING THEORY OF NATURAL SELECTION AND THE SELFISH GENE

Can Yıldızlı

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

August, 2011

INCREASING CHANCES OF SURVIVAL FOR MALWARE USING THEORY OF NATURAL SELECTION AND THE SELFISH GENE

Approved by:

Assoc. Prof. Dr. Albert Levi (Dissertation Supervisor)	.....
Prof. Dr. Bülent Yener (Dissertation Supervisor)	.....
Assoc. Prof. Dr. Erkay Savaş	.....
Assoc. Prof. Dr. Osman Uğur Sezerman	.....
Assoc. Prof .Dr. Yücel Saygın	.....

Date of Approval: .....

INCREASING CHANCES OF SURVIVAL FOR MALWARE USING THEORY OF NATURAL SELECTION AND THE SELFISH GENE

Can Yıldızlı

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, 2011

Thesis Supervisors: Albert Levi & Bülent Yener

Keywords: Zararlı yazılımlar, botnet, bencil gen, doğal seçilim

Özet

Zararlı yazılım terimi genel olarak bilgisayar virüsleri , truva atları, kurtçuklar ve diğer zarar verici program veya kodu belirtmek için kullanılır. Zararlı yazılımı kodlayan kişiler antivirüslerin bulamaması için kodlarını gizlemeye çalışırlar. Antivirüsler şifreleme ve gizleme yöntemlerini bulabilmek için değişik teknikler kullanmaktadırlar. Zararlı yazılımın başka bilgisayarlara bulaşabilmesi, bulaştığı makinanın kaynaklarını kullanması ve kendi kopyasını çıkarabilmesi için hayatta kalması saldırganın başlıca ilgilendiği konudur.

Darwin'in doğal seçilim teorisi ve Richard Dawkins'in bencil gen konseptinden yola çıkarak zararlı yazılımın hayatta kalma şansını arttıracak yeni yöntemler anlatılmıştır. Bencillik, fedakâr davranış, taklitcilik, grup seçilimi ve benzer davranış modelleri denek zararlı yazılımımıza eklenmiştir ve önerilen teknikler mevcut çözümlere karşı test edilmiştir. Bu tezde gösterilen özellikler ile zararlı yazılımı geliştirmek için yardımcı bir araç yazılmıştır. Önerilen tekniklerin etkisi gösterilmiştir ve 300.000 üzerinde zararlı yazılım örneği ile deney gerçekleştirilmiştir. Grup davranış modelleri tanıtılmıştır ve botnetleri geliştirip daha sağlam hale getirmek için yöntemler önerilmiştir.

INCREASING CHANCES OF SURVIVAL FOR MALWARE USING THEORY OF NATURAL SELECTION AND THE SELFISH GENE

Can Yıldızlı

Computer Science and Engineering, Master's Thesis, 2011

Thesis Supervisors: Albert Levi & Bülent Yener

Keywords: Malware, botnet, selfish gene, natural selection

Abstract

Malware, short for malicious software, is used as a general term for computer viruses, Trojan horses, worms, and other harmful software or code. Malware authors try to obfuscate their code in order to evade antiviral programs. Different analysis techniques are used by antiviral programs in order to detect different encryption and obfuscation methods. Survivability of malware becomes the main concern for an attacker since the malware should usually be able to spread to other computers; use resources of victim's computer; and create new copies of itself.

In this thesis, inspired by Darwin's theory of natural selection and the selfish gene concept explained by Richard Dawkins, we propose novel methods which increase the chance of survivability for malware. We implement selfishness, altruistic behavior, mimicry, group selection, and similar behavior models into our experimental malware and we also test our techniques against existing solutions. We develop tools in order to enhance existing malware with features presented in this thesis. Effectiveness of proposed techniques are presented and an experimental test is carried out with a dataset containing more than 300.000 malware samples. Group behavior models are also introduced and methods proposed for enhancing botnets to have better stability (Evolutionarily stable botnet).

Dedication

To my love, Damla TOKER, for always being there for me.

And to my parents, Sevgi and Birol, for their many years of selfless sacrifice.

Acknowledgements

First I owe my deepest gratitude to my supervisors, Albert Levi and Bülent Yener, for their continuous support and encouragement. Their guidance helped me during the time of my research and writing of this thesis.

Besides my advisors, I would like to thank my family for supporting me throughout all my life and for their inspiration:my grandparents, Fethiye and Haluk Söylemez, my parents, Sevgi and Birol, and my brother, Uğur.

I also would like to thank everyone who has helped me along the way, especially my friends and my girlfriend.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer viruses are programmed to be stealth, persistent; and they are mostly well crafted to be able to spread rapidly. Developing these viruses is a tedious task and virus authors should be familiar with many areas of computer science such as network protocols, information retrieval, cryptography, distributed systems, artificial intelligence, and even computer vision. Moreover, a virus should generally be as small as possible and should stay hidden in the victim's machine long enough for a successful attack. Virus authors know various advanced methods to obfuscate their code. As a result of this, antiviral programs try to implement quick detection and disinfection methods against those sophisticated viruses. Self-propagating viruses can spread to more computers for more resources and Trojan horses can steal information without being detected for a long time. As antivirus continue to advance, the survival of malware is rapidly becoming the virus authors' main priority.

Malware propagation includes different methods such as polymorphism and metamorphism. This creates different obfuscation methods and produces variants of a malware which have the same functionality. We believe that new propagation techniques will not only change the representation of the code, but help to create new functionality.

Motivated by "the Selfish Gene" concept [12], we believe that there are similar needs for the malware as there are for genes. Computer viruses can be considered as a gene or a meme[4]. We prefer to use the term gene instead of meme since the latter is not

as common a term as the former, but a reader who is familiar with both terms should be convinced that both behave in the same way to survive. However, it should be clear that viruses are computer programs which make it impossible to propagate without hard-coded instructions or user interaction. In addition to that, mutation is another feature for genes which can hardly be observed in malware. Mutation is an error in the copying process. Whenever a program copies itself, there is a small chance to see a disk error or an environmental condition which can change some of the bytes being copied. This error most of the time makes the program useless since there is a high chance for the program to crash. This error may also help a virus to change its signature if the error in the copying process does not effect the function of the program. In our work we will not go over details of the computer virus mutations. We assume that these mutations may happen. Despite the fact that genes are a part of living organisms and computer viruses are just formed by instructions, their purposes are nearly the same. They both try to extend their environment for survival and they both tend to replicate. Instead of genes, we refer to species which are defined as survival machines controlled by their genes.

1.1 Our Contribution

The purpose of this thesis is twofold: first, we propose techniques which can be used by malware to increase their survivability in the light of the behavior of different species; and second, we present how strategies from game theory help to maintain a stable botnet. Proposed techniques for survivability are mostly derived from the actions of the species under a threat such as predators. We try to explain the use of many concepts from the theory of natural selection[11] and “the Selfish Gene”[12]. We compare the survivability of modern malware and our experimental malware which uses these concepts such as selfishness, altruistic behavior, mimicry, group selection, and more. In our work we present BEMWARE, which is our tool that can produce a new virus with the techniques we present from existing viruses. A virus enhanced with the tool will preserve its ancestor’s functionality but will survive much longer. We also carried out an experiment with more than 300.000 malware to evaluate our methods against current antiviral solutions.

1.2 Goals

Our goal is to show similarities between living organisms and malware. From these similarities we present some behavior models of species which can be used in malware development. We call this BEMWARE (Behavioral Enhanced Malicious Software). We show that our techniques can increase survival chance for malware

We defined a list of our arguments for this thesis below.

- We explain behavioral characteristics of evolution such as virus rivalry, viral adaptation, altruism, mimicry, parasitism, mutualist behavior, botnet as a species, and strategies for botnets including ESS (Evolutionary stable strategy), Tit for Tat, and Grim trigger.
- We will not discuss about virus mutations. Mutations can happen during any data transfer and malicious code can change into something new. We also will not discuss whether these mutations can improve the ability of viruses or make them malfunction.
- We do not make use of genetic algorithms in any technique, however we believe that it can support the idea of mutation in malware. Our argument is to use behavior models of species and we do not want to restrict ourselves in one type of evolutionary algorithm for implementation.
- We do not consider polymorphic code or metamorphic code as mutations or evolutionary. These features will not change the function of the malware and they cannot increase the chance of survival [8].
- We will be using our experimental BEMWARE for our test cases for most of the time against common antiviral solutions. Our test dataset contains more than 300.000 malware samples and 10.000 benign samples. We only make use of this dataset for specific tests.

1.3 Challenges of the Field

In this section, we describe the difficulties and problems of our research.

1.3.1 Biology, Philosophy, and Computer Science

Firstly, we are not biologists. Our knowledge and reasoning help us to put some ideas from biological science into computer security field. Secondly, we presented implementations for most of our techniques but for some of the methods we presented, we could not give any real life examples due to diccifulty of their implementations. We support the efficiency of these methods by giving examples from other sciences. One can claim that some of our ideas are based on philosophical assumptions. We want to quote Mark Ludwig's words about the philosophy and the science, and leave the judgement to the reader.

"To be a good scientist, one must also be somewhat of a philosopher. A few centuries ago, scientists were called natural philosophers. There was wisdom in that. When a scientist fails to be a philosopher, he tends to be blinded by philosophy. Then he becomes a slave to what he believes, rather than its master."[21]

1.3.2 Implementation and Platform

We developed most of our techniques in x86 Assembly language. We selected Windows platform for development since the vast majority of malware targets Windows users. We also implemented most of our experimental BEMWARE using C++ language.

1.4 Terminology

Some of the terms we use in this thesis are somewhat different than their original meaning. An example term is the "computer virus". We used the term "malware" to represent computer viruses, but computer viruses need not be used for malicious activity and the term is a generalization for all types of harmful software. In this section, we describe most

frequent terms used in this thesis so reader should not be confused about their meanings.

Malware Malware, short for malicious software, refers to hostile and intrusive programs. The primary objectives of malware consist of gaining unauthorized control over system resources; interrupting, initiating or denying operations; gathering private information and exploiting or corrupting the collected data. The term is mostly used to describe computer viruses, worms, Trojan horses, botnets, adware and spyware.

Altruistic Behavior Altruism is the act of selflessness, where an entity risks its own welfare in order to increase another's. The stinging bees protecting their food stocks against honey robbers is a good example of altruistic behavior. The stinging bee attacks the intruder and sacrifices itself to defend the honey, since the bee loses its vital organs in the act of stinging. Therefore, the bee works for the greater good of the colony, even though it cannot benefit from this situation.

Selfishness Selfishness is exactly the opposite of altruism. Selfish behavior indicates a complete concern with oneself, where the individual consciously harms or impedes others for personal gain. An example of this behavior can be observed in the emperor penguin colonies, in the Antarctic. All penguins hesitate at the edge of the water before diving in, in fear of becoming prey to seals. In order to minimize this threat, a penguin has to be the bait and dive in before the others. Naturally, none of the penguins want to put themselves in risk, leading to a selfish challenge of waiting at the edge of the water and trying to push each other in.

Mimicry Mimicry is a term used in evolutionary biology, which refers to a species imitating certain characteristics of another. This adopted aspect can be in appearance, behavior, noise, smell and habitat, which increases the chance of survival of one or both species.

Parasitism Parasitism represents a type of relationship between two species, in which the parasite takes advantage of the host at the expense of the latter. Parasites benefit from the hosts' resources to advance its chances of survival.

Tit for Tat Tit for tat is an expression that stands for retaliation in response to an injury from another. The term is often used as a game strategy in "The Prisoner's Dilemma", where the concept of equal retaliation results in the most rewarding outcome, regarding the possibility of the opponents deceit.

Grim Trigger Grim trigger is a game strategy that can be adopted in a repeated game such as "The Prisoner's Dilemma". In this approach, the player cooperates until the opponent defects. Once the player realizes the betrayal, the trigger condition is satisfied, which results in the player defecting for the remainder of the game.

Virus A virus is a computer program that is designed to interfere with the operation of the system and the data on a computer. Viruses can replicate themselves and infect a computer before spreading to other devices on a network or through a removable medium such as USB drives.

Mutual relation Mutual relation, also known as mutualism, is the relationship between two species, in which both individuals benefit from the relationship and sometimes even depend on it for survival.

Polymorphic Code Polymorphic code is code that mutates and changes each time it is activated, without altering its functionality.

Metamorphic Code Metamorphic code is code that can reprogram itself. As a result, everything within the program changes except for the actual functionality, so that the children will never look like their parents.

Static Analysis Static program analysis is the process of analyzing a computer software without executing the program itself. This process is faster, but it can be easily evaded by malware. Static analysis makes use of signature matching and flow analysis graphs.

Dynamic Analysis Dynamic analysis is the process of analyzing a program by executing it on an emulator, and watching its execution behavior. This technique is slower than static analysis but necessary to examine the obfuscated code. An obfuscated code or an encrypted code should be restored to its original form in order to be executed. Dynamic analysis helps to detect these behavioral abnormalities and decrypted code.

Binary The term binary refers to executable, i.e. a type of file that contains a code for the computer.

Trojan Horse A Trojan Horse is a malicious program that is disguised as an application. This program enables the attacker to access the victim's computer without his permission or knowledge. A Trojan Horse does not replicate or copy itself, but still compromises a computer's security.

Crypter Crypter, also known as scrambler, is a utility that modifies an executable file and is often used to obfuscate the code. When the obfuscated executable is initiated, the code expands to the original form. Malware authors usually use crypters to avoid virus detections.

Evolutionarily Stable Strategy Evolutionarily stable strategy, or ESS, is a strategy which, if adopted by a majority of a population, cannot be overridden by an alternative strategy that is initially rare.

Botnet A botnet is a network of infected and compromised computers, referred to as bots or zombies, that are used for malicious purposes without the owners' knowledge or permission.

Replicator The Replicator is a molecule that is able to make copies of itself.

1.5 Structure of the thesis

Outline of the rest of the thesis is as follows: In Chapter 2, we present related work about evolution and malware. We explain how our work differs from the existing ones. In Chapter 3, we explain behavior models for species and describe how malware can benefit from those models. We give examples to show similar implementations of these methods to enhance malware. Then, in Chapter 4, we present different strategy models and apply these models into botnet to increase stability. We use common strategies such as Tit for Tat, ESS, and Grim Trigger from game theory. In Chapter 5, we present an experimental malware(BEMWARE) which uses some practical approaches to present our techniques. Experiments and results are also presented in this chapter. Finally in Chapter 6, we conclude the thesis by providing an overlook and give some ideas for future research.

Chapter 2

Related Work

There are a few similar topics which are related with our subject. A comprehensive work is published by Mark Ludwig in his little book of computer viruses, namely "Computer Viruses, Artificial Life and Evolution" [21]. He describes why it is important to study viruses and try to show the links between life and computer viruses. He questions some philosophical ideas such as "Are viruses alive?", the importance of philosophy and its relation with science, mechanical properties of life, and he describes the emergent behavior which opens a debate about viruses as a deterministic living organism. He mainly describes mutation and explains this in mathematical terms. Our research,however,does not try to answer whether a mutation can exist or not. We support the idea of studying viruses so that we can learn more things about the life itself. Ludwig does not describe relations and similarities between viruses and behavior model of species. Instead, he tries to answer some important concepts such as artificial life, virus mutations, and incalculability.

Another research which is related to malware survival is explained in the book "Malicious Cryptography: Exposing Cryptovirology" by A.Young and M.Yung [32]. They describe how to use elements of game theory and cryptography to create a survivable malware. Cryptovirology is a field which combines cryptography and virology. The field studies advantages of using cryptography for malicious intent. In their book A.Young and M.Yung proposes methods to enhance computer viruses to use strategies which help them to gain opportunities against humans and disinfection. In Chapter 4, we present some sim-

ilar methods for botnets which are derived from group behavior and game theory. Most of our techniques can also be considered as practical examples of cryptovirology.

In [27] H. Spafford defines computer virus generations, virus metabolism, and presents his idea of artificial life. It is also the only work which mentions the individual virus behavior. He gives the example of territorial behavior, predatory behavior, and self-protective behavior. In his paper, he also notes that virus characteristics come from its author and living organisms generally change without any obvious exterior agency unlike viruses. His work supports our ideas about using behavior to understand more complex interactions between viruses and clarifies the concept of malware mutation.

Evolution in malware is also discussed in [28]. Authors define a model for evolution using API selection from environment. They also define Clean-application mimicry and explain how new APIs can be beneficial for malware to evade antiviral solutions. Authors also give example from biology and explain relations between biological viruses and malware. In Chapter 5 we give a runtime crypter as a Mimicry example. However, we define evolution in terms of gaining new functionality later in that chapter. Our mimicry example is given to clarify the subject, not to present an evolutional change.

Malware detection is closely related to survivability. In [7], authors show that there is no algorithm that can perfectly detect all viruses by using notions and Cohen's existing work [9]. Authors also show that even if we have a sample of a virus, we cannot write a program which detects that particular virus without any false positives.

In [2], authors define evolution and discuss the fittest malware in terms of their detection level. Antiviral solutions shape the surviving viral population by deleting the unfit individuals. Authors discuss malware mutation and they group evolution into two classes. Mutations can occur from environmental changes or in the virus body itself, which is defined as somatic changes. Authors claim that environmental changes may occur due to a number of factors such as compilation of different compilers and somatic changes can be observed in accidental disruption of virus code or mutation. In this paper, mutation is explained as an outcome of an accidental event such as system crash, corrupted disk, electromagnetic radiation, faults in hardware or storage media.

There is another publication which proposes genetic algorithms for malware evolution [25]. Authors extracted features from Bagle worm and used standard genetic algorithm to produce new variants of it. We believe that the functionality of the malware should also be changed in order to call it an evolution which makes this work an extended version of polymorphism. However, it is the first example of using a genetic algorithm on malware which is worth mentioning.

As far as we know there are no documented practical methods to include behavior in malware. In this thesis, we propose behavior models to improve malware survival. We implement different tools and develop experimental malware to test the efficiency of our ideas. We explain different strategy models for botnets agaisnt their rivals. We also propose a practical approach for malware evolution with genetic algorithms. We believe that our work is the first step in a series of novel experiments and research directions in computer security.

Chapter 3

Behavior Models

Think about an environment in which there is enough food for a single predator to be fully fed. If a secondary predator comes to the same habitat, a competition for resource starts to emerge. Dilemma for the predator is either to defend the resource by facing the risk of death or search for other resources. It is not an easy task for the predator to calculate the utility of attacking the other predator. We should focus on the purpose of the predator and define consciousness before making an approximation of its action. We will not be discussing consciousness in this thesis. Instead we will try to calculate the utility for viruses using different behavior models.

3.1 Virus as an individual

A malware author generally has a motive. Malware can be designed to spread to as many computers as they can, target a single machine to extract information, or can be used as a tool of hacktivism for a socially motivated purpose. Malware author's role is to teach the malware how to reach its goal by coding functions for all encounters. Slammer worm which infected about 75.000 machines within 10 minutes in 2003 did not have any purpose of being stealth and it did not contain any malicious code against its host[23]. The purpose of Slammer worm was to infect all possible computers by using an unpatched vulnerability. It was designed as small as possible to achieve this goal. The author of the worm actually made a mistake when developing pseudorandom generator module.

When trying to convert a number into negative, author used a “sub” instruction instead of “add”. This error affected spreading speed of the worm since ip scan algorithm is based on this random number generator. The opcode for add eax,1 is 83 C0 01 in x86 assembly instruction set and sub eax,1 is defined as 83 E8 01. These two instructions only differ by 1 byte (C0 for add, E8 for sub). During the spreading process of the worm, a packet corruption or a disk error can change C0 into E8, thus putting more machines in danger and creating more hosts for the worm’s survival. Can this error be considered as a mutation? We leave this question to our reader and continue to explain the purpose of malware.

If we consider a virus as the predator and the internet as its habitat, we can think of actions which give maximum utility for the virus whenever it encounters another predator. Think of a virus in a machine which uses resources to stay persistent for a long time. If it detects that an antivirus is trying to be installed, it can either try to stop the installation process or try to spread to other computers by using more resources before the installation completes. Stopping the installation might increase the user’s suspicion about a virus existence in the system and may lead to attempting to install other antiviruses or a complete format. On the contrary, user may not want to spend more time trying more solutions and gives up the idea of installing an antivirus. The efficiency of this virus is based on its choices for a given set of purposes.

3.2 Virus adaptation

Modern viruses are adaptive and they have protection mechanisms from common antiviral solutions. What we mean by adaptive is they can sense the presence of an antivirus or other viruses which exist in the same host. They can also detect the capability and size of the resources that they can use to reach their purpose. A good example is a hybrid botnet in which individual bots should detect if the host is behind a NAT or not [31]. This helps the botnet to form a better network structure to reach its goal. We will be explaining group behavior later in this chapter.

Adaptation also creates an opportunity for any problem that viruses may encounter.

A virus which operates when the CPU is idle is less suspicious from the user level than where the resources are allocated for something else. They may also use peripherals of a computer to detect the surroundings. This includes listening around, checking from webcam to see if anyone is using the computer, or even detecting the temperature for a possible CPU overclock to increase its resources.

3.3 Parasitism and Mutualism

Adaptation was a primal feature that should be inherited by malware but surely is not enough for its survival. If a malware needs to stay hidden and transfer itself to other hosts, it may need different behavior models. One example is parasitism. The detection ratio for a malware increases whenever it tries to interact with its environment. The host in which the virus resides in might also be so paranoid that it only allows certain applications for communicating with other hosts. A virus in this environment can make use of the existing software in the host to spread to other computers and protect its significance in the gene pool. We developed a proof of concept code of such a virus where it only infects the programs which use WinSock library for communication. The virus uses direct code injection and hooks send() and recv() functions of the host program. The virus starts to monitor outgoing packets and whenever a PE(Portable Executable) header or a ZIP header is detected, it sends its own code instead of the original packet. The virus is 5,192 bytes in size and developed in x86 assembly language using Masm32 compiler.

Mutualism can also be used to stay resident for a long time. Suppose a virus used a vulnerability of the operating system or a program to propagate. It can then update the operating system and programs frequently so that no other predator can compete with itself. This is good from the user level since it will be harder for other malware to compromise the system and it is beneficial for the virus since all the resources belong to itself. Viruses developed without any malicious intent can also be used to patch vulnerable systems.

3.4 Mimicry

Mimicry is one of the most used behavior models for survival by modern malware. We can see many real life examples to evade antiviral solutions and trick users to believe that they are in fact benign programs. One famous mimicry behavior can be observed in the FakeAV virus [1]. The virus disguises itself as a legitimate antivirus and warns the user of a possible detection and prompts them to download itself. FakeAV virus also contains such properly designed GUI and informational messages that even an advanced user can fall for the trick of this mimicry. Another form of mimicry is to trick antiviral solutions so that they believe the virus is a benign application. Implementation of this feature is more complicated since FakeAV only tries to trick users whereas this technique is used to evade a predator which is programmed to delete malware. We explain the details of implementation in Chapter 5.

3.5 Altruistic Behavior

Viruses do not generally have distinct purposes and children inherit this purpose from parents most of the time. An individual virus can act in order to help other viruses to achieve this global goal. Altruistic behavior, just like in nature[16], can help a family of viruses to survive longer. This behavior can be perfectly examined in a botnet since each member of the botnet can be considered as an individual of the same species. Antiviruses update their definitions frequently and it becomes a real challenge for a virus to keep control over its host. Another disadvantage for the virus is the number of antiviruses and different signature databases they use. Whenever a new threat is detected either by static or dynamic analysis,the user is notified with a warning from the antivirus and multiple choices will be presented to user including deletion, disinfection, and quarantine. If a member of a botnet is detected by an antivirus or even found as suspicious, there is a huge chance that it will be sent to the antivirus company for further analysis. This process is necessary for any antivirus company to update their signature database and helps them to

[1] http://www.symantec.com/business/security_response/writeup.jsp?docid=2007-101013-3606-99

implement new detection techniques based on trending threats. It is also beneficial for an antivirus company to share their signature database with other companies. This will result in losing more members of a botnet within short period.

Some techniques were used to prevent the user from deleting malware. One example is to create other processes to watch each others' back. In this way, whenever one of these processes is terminated, others try to initialize it again by downloading from a fixed source or constructing it from the payload that they carry. Antiviruses did not have any difficulties to counter this strategy. Botnets need an efficient method to prevent losing control over the hosts and we claim that altruistic behavior can help them to survive longer.

Imagine a variant of a bot which is detected by an antivirus as a new threat. The bot can change its definition to a known virus signature by analyzing that its being scanned by hooking antivirus itself. An example of such a bot is developed as a proof of concept. This signature will make the bot look like it belongs to a known malware family and antivirus will not try to examine the sample since it is already known. Sacrificing its own life, the bot probably saved many variants of itself on other hosts. Another desirable action for the bot is the ability to delete itself completely instead of using mimicry to look like another virus. This behavior is harder to implement but it is also another use of altruistic behavior.

Another example is to have a virus coded as a node of a linked list. The node is the virus code itself. The previous node pointer actually contains the difference from its parent's code in encrypted form. Whenever the virus creates a new variant, it also encrypts the new differences from its child as its next node pointer. If the viruses are adaptive, they can detect which nodes are missing due to detection and they can analyze what changes triggered the detection. Knowing this will help the remaining viruses to create new variants based on what they've learned from their loses.

3.6 Virus Rivalry

Virus rivalry is a common situation when two or more parties start to use same resources. This environment causes a long term competition.

There are two ways in which a virus can win this competition. Firstly, a virus can start

a fight and develop detection and removal procedures against its rival. It should be noted that any feature malware inherits is susceptible for detection by antiviral solutions. What we recommend here is not armoring malware with numerous codes against all enemies but to develop a behavioral algorithm which changes its actions based on its rivals properties. A famous game "Core War" is a nice example in which rival programs compete for survival on the same memory [13]. The game simulates a shared memory and executes two programs by running them on random locations. Each battle program is coded in an abstract assembly language similar to x86 Assembly and executed by a virtual machine called MARS (Memory Array Redcode Simulator) [10]. Both programs have their own instruction pointers but they do not have any knowledge of their location on memory since the memory address is wrapped around. They also do not have any clue about their rival's location and they should somehow use the most dominant strategy to survive.

The game represents a real world scenario exactly for a virus competition.[2] A more complicated situation occurs when antivirus also starts competing against the viruses. This gives the opportunity for a virus to use a second way to win the competition. It triggers the antivirus or creates some diversion for the user like opening some popups, making him suspicious about his computer's integrity. This behavior can be seen in jackals. When a pack of hyenas feed on a prey, they do not share their trophy with other animals. Knowing this,the jackal shouts and calls the lions to turn their enemies against each other. It then waits for a right time to jump in and take a piece from the prey. In our case, if the virus does not contain any clever solution against a competing virus, it can then use other fighting strategies such as initiating a scan [3], or notifying the user about other viruses' actions or place.

[2]One can claim that Windows operating system reserves isolated memory for each process. However, a virus can detect other viruses' exact location on RAM

[3]The virus must be sure that it is not detected by the antivirus which it asked for help

Chapter 4

Botnet strategies

We discussed the behavior of viruses, but most of the time, viruses spread to other computers and form a network in which they can communicate. This structure is called a botnet and the individual viruses are referred as bots.

Bots generally inherit the same behavior models since spreading does not change their functionality. They use different methods to communicate with each other. When we are talking about botnets we should also mention that their purpose can change dynamically. Current botnets do not have distinct behavior models for different purposes. Basically, bots can be classified based on their connection protocols and topology [33]. Mainly, there are centralized botnets which use a server to issue commands and control all connected bots to that server. These are decentralized or commonly known as P2P botnets which are formed by bots connected to each other. In order to command this type of botnet, an attacker can issue a command from any bot belonging the network. There is also a hybrid model which is mentioned in [31] but does not have any real life examples yet.

It will be helpful to imagine the botnet as a secret organization with members who know each other but try to protect their identities and purpose from the outer world. The purpose of this organization is mainly earning money, making a political statement, or to sabotage. There are also some organizations which are formed just for fun. We discuss the strategies used by the members of the botnet in the following sections. We give example of a botnet which uses a poor strategy. Then, we propose another strategy which has a

higher utility in most cases. At the end of the chapter, we discuss the possible expected state of botnets.

4.1 Grim Trigger

Grim trigger is a strategy for a repeated game where a player starts defecting as soon as the opponent refuses cooperation [22]. We wanted to explain this strategy since it is used in a widely known botnet called SpyEye even if it performs poorly.

4.1.1 Zeus vs SpyEye

Zeus is a widely known banking malware also known as ZBot [3]. It is actually a centralized botnet which has a main purpose of getting the victims' credit card numbers and bank login information. ZBot was identified at the end of July 2007 and become popular in 2009 with continuous updates and modules by its authors. SpyEye is a similar bot which is known as ZBot's rival. The reason for calling it as a rival is that it uses a specialized strategy against Zbot. It scans the processes until it locates ZBot's presence, and then terminates all threads of ZBot, cleaning the ZBot binary afterwards[5].

Aggression of SpyEye did not help much in terms of propagation since ZBot does not prevent any communication of other processes. It only helped SpyEye to use more resources of the compromised host. We can also say that SpyEye became much more popular because of the "Kill ZBot" module and many malware researchers began to analyze it giving information to antivirus vendors. The strategy for the SpyEye did not help the purpose of the botnet at all.

Things started to get complex when ZBot authors implemented an anti-SpyEye module into ZBot. The consequence of the implementation is interesting for our research. After a short while, Zeus and SpyEye both lost their significance and ZBot source was given to SpyEye author, then leaked into public.[1] This is a good example to show that the grim trigger strategy is a poor strategy when the purpose is to stay longer on many

[1] You can download it from www.cryptovirology.org

computers. It should be noted that the actions of these bots are limited by their authors since they pick a strategy and implement it into them. This somehow opposes our idea of behavior enhanced malware since what we want to achieve is an individual malware, which can change its strategy against any encounters based on its purpose.

4.2 Tit for Tat

In order to understand the efficiency of an aggressive strategy like the one SpyEye uses, we should focus on "Prisoner's Dilemma". It is a simple gambling game played by two people. Although it has many versions, it is better to explain it using its original form. Two suspects are arrested and questioned by the police. They cannot communicate with each other and both have two options to choose from (testifying against the other or remaining silent). If one testifies against the other (Defects) and the other remains silent (Cooperates), the defector goes free and the other will be sentenced for one-year in prison. If both remain silent, they will be sentenced to one month in prison. Instead of remaining silent, they can both testify against each other. In this situation both suspects will be sentenced for 3 month in prison. Each prisoner must choose either to testify against other or remain silent.

If you play this game, you will realize that always defecting will be the only strategy that gives you highest payoff. However, it is known by both parties that if both cooperate, their individual reward will be greater.This is why the game is called a dilemma.

In this version of the game, there is no way of ensuring trust. If we repeat the game a couple of times with the same players and letting them choose based on their past experiences, they may start forming some kind of trust. There are different strategies for the iterated version of Prisoner's dilemma but we will focus on tit for tat strategy which initially cooperates then chooses to cooperate or defect based on the opponents's previous action.

Tit for tat strategy is an effective strategy for our case [14]. Suppose that a bot and its rival coexist in a host. If both bots use the resources of the computer by defending, they both gain 30$ per day. If one of them attacks the other and successfully prevents it from

accessing the resources, then the attacker will gain 50$ per day and leaves the other party with no money. If both attack and try to prevent each other, they will both gain a sum of 10$ per day.

This is an example of a Prisoner's dilemma and we can examine the efficiency of a SpyEye, ZBot and two other botnets (We call them TFT1 and TFT2 for this example) which play tit for tat strategy. We assume that SpyEye and ZBot always attack other bots and try to get a better payoff.

Suppose that the game lasts for 10 days and each day is an iteration of the game which has a scoring as described in Table 4.1.

Table 4.1: Payoff table for bots

	Defend	**Attack**
Defend	30,30	0,50
Attack	50,0	10,10

Also, suppose that these bots coexist with each other in different machines, making use of the resources to gain money for their authors. Let's calculate each bot's 10 day earnings.

The SpyEye bot which coexists with TFT1 will earn 50$ for the first day. TFT1 plays the tit for tat strategy and it will attack for the rest of the game so SpyEye will earn a total of 50+9*10 = 140$ whereas TFT1 earns a total of 90$

On the second host SpyEye bot coexists with TFT2 and again the same amount of money is collected by SpyEye bot in 10 days. TFT2 earns 90$ since first day it defended SpyEye's attack and started attacking for the rest of the game due to tit for tat strategy.

Lastly SpyEye bot coexist with ZBot. Since both will try to attack each other, they will both earn 100$ in 10 days.

SpyEye collected 140 + 140 + 100 = 380$ in 10 days with its aggressive strategy.

TFT1 earned 90$ from the host which is shared with SpyEye and another 90$ from the machine of ZBot. When TFT1 coexist with TFT2, since they both defend for the whole game, both will earn 300$. The sum of TFT1's individual earning is 480$ which is greater than the SpyEye's earnings. Table 4.2 lists sum of earnings.

Table 4.2: Total earnings of bots.

Bot Name	**Total earnings**
SpyEye	380$
ZBot	380$
TFT1	480$
TFT2	480$

Table 4.3: Modified payoff table for bots

	Defend	**Attack**
Defend	30,30	0,200
Attack	200,0	10,10

If we change the payoffs and make it more advantageous for the aggressive strategy as in Table 4.3 and increase the game duration to 1 month, then we can still observe tit for tat strategy is dominant over the others. Total earnings for the bots can be seen in Table 4.4.

Table 4.4: Total earnings of bots for modified payoff table

Bot Name	**Total earnings**
SpyEye	1280$
ZBot	1280$
TFT1	1480$
TFT2	1480$

The tit for tat strategy is mostly preferred when the most of the bots in the population tend to defend.

4.3 Evolutionarily Stable Strategy

In the previous section, we described that tit for tat strategy results in a better payoff in the longer run if the majority of the bots chooses to defend. In an environment where all bots try to prevent each other from using resources, it may not be feasible to use a tit for tat strategy. So the question is: Which of these strategies will overrun others?

Let us define the environment, rules, and behavior models which coexist in our model. The first rule is whenever two bots compromise a host, they will have two options. They can either choose to attack the other bot, or they can try to avoid the attack. Suppose we have two distinct types of bots. ZBot variants and replicators. Replicators are stealthy survivors which cannot be detected easily. They do not have a fighting strategy other than attacking random memory locations. On the other hand, ZBot variants uses aggression against other ZBot variants and replicators. It quickly develops a cleaning routine and deletes the rival malware. If two ZBots attack each other, one of them will survive and the other will use the remaining resources on the host. If a ZBot variant fights with a replicator,the replicator will quickly back off and spread to the other computer. If both replicators meet each other, they will try to eliminate each other by writing to random memory locations.

We assign scores based on their success of obtaining resources like the following:

- If a ZBot deletes a rival it will gain 50 points.
- If a ZBot is deleted by its rival, -100 points will be given as a punishment.
- Replicator will gain 50 points if it can win any fight
- If both replicators fight, one of them will back off after being found by its rival. We will give a score of -10 for both replicators since they used time and resources to attack each other.

This example is a modified version of hawk-dove game from “the Selfish Gene” book [12] and it is given to describe evolutionary(or evolutionarily) stable strategy. Evolutionary stable strategy is a strategy which, if adopted by a majority of a population, cannot be

overridden by an alternative strategy[30]. For our example, lets consider all the hosts that contain only replicators. We can expect that a replicator wins half of the fights since both of them uses a random strategy in order to get rid of the other. This results 40 points for winning (50 points winning and -10 penalty for using a lot of time and resources) and -10 points for losing. We can also say that a replicator gets 15 points on average. This can be considered as a real life example since bots do not have many strategies against their rivals. Now suppose that a ZBot variant joins as an alternative to replicators. ZBot will quickly start dominating replicators since replicators will leave the host whenever ZBot tries to attack them. This results in a huge advantage for ZBot variant since its avarage payoff will be 50 points. ZBot variants quickly begin to take over hosts.

Now imagine all hosts are compromised with ZBot variants. If a ZBot spreads to a host and starts a fight with another ZBot, one of them will be deleted. Winner will get all the resources of the system which is represented as 50 points whereas deleted ZBot gets -100 points. ZBot can expect to win half of his battles and on average ZBot will get -25 points. Now suppose that a replicator arises in the network. Its average payoff will be 0 since it does not win any fights. This average payoff is better than ZBot variants' which help replicators to spread quickly.

It will be naive to expect that this network will always change between ZBot and replicator domination. There is a stable ratio of Zbots to replicators. Whenever this ratio is reached, the average payoff for both malware will be equal. A good property of this stable state is that no other strategy is able to dominate it in the long-term since an individual is liable to act with a better average payoff, thus forming a new EES. Botnets can maximize their payoff with a dynamic strategy based on discussed average payoff.

Chapter 5

BEMWARE

So far we described behavior models and strategies for malware. In this chapter, we will start from the techniques to evade antiviral solutions and present methods which can change the functionality of the malware. The important thing is that we are not interested in any form of polymorphism[24] or metamorphism [18]. These are well known and implemented techniques to generate new variants which have different signature but same functionality. Since variants cannot be considered as evolution we should focus on how to change the purpose and methods of the malware. We use the term "BEMWARE" to describe a malicious software which uses behavioral algorithms or evolutionary behavior in order to survive longer. We first present a mimicry behavior for malware which enable to bypass many protection mechanisms.

5.1 Mimicry implementation

In section 3.4 we explained how mimicry can increase malware survivability. A common example is a crypter which can enhance malware with mimicry feature to evade antiviral solutions. We will explain how a crypter works and present parts from our implementation. Our experimental crypter currently produces a representation of any malware without changing their functionality and the output becomes undetectable by all antiviruses we tested.

Crypters are programs which encrypt the code of the malware [6]. This encryption

helps them to bypass signature detection of antiviruses. This method is extremely effective against static analysis, since antivirus tries to match the patterns in the malware with the ones in its database. Dynamic analysis, on the other hand, executes the code in an emulator and tries to detect a malicious code or behavior. Based on the fact that an encrypted code should be decrypted in order to be executed, dynamic analysis easily detects the malicious code once it is decrypted in the emulated environment [20]. Next section describes techniques to prevent dynamic analysis from detecting the code by changing properties of the malware and showing it as a benign file.

5.1.1 Inner workings of a Crypter

We explained that a crypter encrypts the malicious code but we did not elaborate on this topic. We already mentioned that an encrypted code needs to be decrypted. Crypter adds this encrypted data to a program which is responsible for the decryption. We will call this program "Stub". Stub is a regular program which,upon execution, decrypts the data attached to it and executes whatever is stored in that data. If the stub tries to write this decrypted (original) form of malware into the disk and attempts to execute it, antiviruses will detect its presence since they observe essential APIs such as CloseHandle() and CreateFile(). This is why we need a way to execute the decrypted code on memory.

The difficulty of executing raw program data from memory is caused by the Windows operating system design. Each binary in Windows contains a PE (Portable Executable) header. This header contains information about the binary execution details such as position and size of its code section, data section, functions of the program, and resources it carries. Whenever a program is executed, PE header is parsed by the operating system and all registers, memory sections, and libraries are arranged based on this header. How could it be possible to execute a program code which is in memory then?

Most programs use APIs (Application Programming Interface) in order to interact with the operating system. Crypter uses some specific APIs in order to execute the malware code from memory. Now we briefly explain the steps for our crypter and give technical description later.

- Crypter takes a malware as an input. It reorders its bytes by randomly switching each byte position with the other. In this way, it evades any form of static analysis.
- A small graphic file is constructed. Shuffled malicious code is added at the end of this graphic file.
- Crypter generates an executable called “Stub”. The stub is responsible for the reconstruction of the malicious code in memory and execution. Stub code does not have any signature detected by antiviruses.
- Crypter then adds the graphic file which contains the shuffled malicious code as a graphic resource into the stub.
- Stub is ready for execution. A smaller stub size increases propagation speed for malware

Crypter can process new malware and adds it into a new stub each time. Crypter is a program used by the attacker so that the application, which will be distributed to the victims, is the stub with encrypted resource.Now we define the execution steps of the Stub.

- Stub uses some anti-emulator tricks and tries to force any antiviral solution to check for different execution paths.
- Graphic resource attached to stub is filled in memory. Since Stub already knows that a graphic file is added, it quickly finds the actual shuffled malware code.
- Stub reconstructs the file on memory. Since antiviruses already stop searching for the real execution path, they will try to detect any malicious behavior from now on.
- Reconstructed file on memory has a valid PE section and code. It is not possible to execute the code directly without letting the operating system read the PE section.
- Stub executes a benign program such as Internet Explorer or it may even execute self. This execution is different than normal execution. Benign program executed

by Stub is suspended and its memory is reachable by the Stub. The benign program will only resume its execution with a command from the Stub.

- Stub reads the PE header of the malicious program in memory and makes all necessary changes in the benign program's execution flow.

We implement the Crypter in C++ and the Stub in x86 Assembly language. We will give technical description of both programs in the following section.

5.1.2 Technical details

We list some of the code that Crypter and Stub use. The explanation of the code will also be given afterwards.

5.1.2.1 Crypter Implementation

The code in 5.1 enables us to randomly shuffle payload(malware) code. It should be noted that the seed for the random number generator is already known by both the Crypter and Stub. In this way, Stub knows how to reconstruct the code.

```
fseek(payload,0,SEEK_END);
payloadsize = ftell(payload);
fseek(payload,0,SEEK_SET);
BYTE * buffer = (BYTE *) malloc(payloadsize);
result = fread(buffer,1,payloadsize,payload);

for (int i = payloadsize-1; i >= 0; i--)
{
    randpos = rand() % payloadsize;
    temp = buffer[randpos];
    buffer[randpos] = buffer[i];
    buffer[i] = temp;
}
```

Figure 5.1: Random shuffling of payload

An image is constructed using the BMP header in 5.2. The shuffled malicious code will look like a legitimate image. It will start after 174 bytes of a valid bmp header+data.

```
bmpfile_magic newval;
newval.magic[0] = 'B';
newval.magic[1] = 'M';

bmpfile_header newheader;
newheader.bmp_offset = 54;
newheader.creator1 = 0;
newheader.creator2 = 0;
newheader.filesz = 54 + totalsize;

BITMAPINFOHEADER newdibheader;
newdibheader.biSize = 40;
newdibheader.biWidth = 10;
newdibheader.biHeight =  10;
newdibheader.biPlanes = 1;
newdibheader.biBitCount = 24;
newdibheader.biCompression = 0;
newdibheader.biSizeImage = totalsize;
newdibheader.biXPelsPerMeter = 8;
newdibheader.biYPelsPerMeter = 0;
newdibheader.biClrUsed = 0;
newdibheader.biClrImportant = 0;
```

Figure 5.2: BMP header for the payload

Constructed buffer is added into the Stub program which is given as the argv[2] 5.3.

Stub is ready to be executed. In the following section we describe the technical details for our Stub and show how it bypasses most prevention systems.

5.1.2.2 Stub Implementation

Most of the Stub contains an anti-emulator code in order to evade dynamic analysis methods. Antiviruses try to detect APIs which show malicious behavior. In our Stub, we also use a method which executes functions of the operating system by calculating their addresses in runtime. Stub is coded in x86 Assembly language since size of the Stub is

```
HANDLE stub = BeginUpdateResource(argv[2],false);

UpdateResource(stub,RT_BITMAP,chunkoff,
MAKELANGID(LANG_NEUTRAL, SUBLANG_NEUTRAL),
allbuffer,payloadsize+174);

result = EndUpdateResource(stub,false);
```

Figure 5.3: Appending payload data into the Stub

important for malware propagation and also helps to avoid signature matching. Fistly we execute the actual Stub code by creating a thread. 5.4 This helps to avoid some antiviruses since they cannot emulate threads. System waits for 5.5 seconds for the Stub code to complete. Usually Stub code completes within 1 second. The reason for using longer time intervals is to evade some antivirus detection.

```
invoke Sleep,1500
invoke CreateThread,0,0,RunSecure,0,0,0
invoke Sleep,5500
```

Figure 5.4: Creating a thread to bypass some dynamic analysis methods

Timing tricks are essential in order to evade emulators like many other techniques [15]. This trick uses the assumption that emulators are slower than real cpu, so a malware can detect if its own code is running fast or slow. Below, our Stub is making two consecutive calls to GetSystemTime and measuring the time difference. If there exists a time difference, it quickly terminates instead of executing the rest of the code 5.5.

We then access the our resource and reconstruct malware code. Reconstruction step is easy since it uses the same random number generator with the same seed 5.6.

The rest of the execution will contain the following path. Stub will execute a benign program with CreateProcessW API in suspended mode. It allocates the memory of this process with VirtualAllocEx. Stub iterates through PE header of the reconstructed file and uses NtWriteVirtualMemory API to change the code of the benign program with the malware. It then uses NtResumeThread API to give the control to benign program which

```
mov ebx, 320B9566h ; GetSystemTime
mov edx, KernelStart
invoke FindAPIn
mov Function,eax ; CALL GetSystemTime

push OFFSET nowS

call Function ; CALL GetSystemTime
sub cx, nowS.wMilliseconds

jz bypassed
fail:
invoke ExitProcess,0

bypassed:
```

Figure 5.5: Timing trick example

actually contains the malicious code.

We test our program on a well known Trojan horse called Poison Ivy and two other bots. Table 5.1 shows the detection rate before crypting and Table 5.2 shows after they are crypted by our tool. Results are based on both static and dynamic analysis.

5.2 Evolving malware

In previous section, we defined API as a function of the program. Applications use APIs for various purposes such as drawing windows, interacting with the user, compression, modifying files and more. In short, APIs create a link between the operating system and programs. We believe that it is the API which defines the functionality of a program. It is also true that APIs are a group of some low level instructions but we choose to explain our ideas in a higher abstract level. Our application mimicry is a built-in feature so it cannot be considered as a functionality gained from a mutation.

This section completely focuses on malware evolution. Artificial life researchers have studied self-replication since 1979 [19] [1]. There is no example of a malware which can increase its functionality when spreading to other computers. Our research shows that it

```
invoke GetResource
add pMemory,174 ;pointer to buffer is at offset 174
sub dwSize,174 ;subtracted 174 bytes from total size

xor eax,eax
xor ecx,ecx
over:
invoke RandomNumber
xor eax,eax
mov ax,RNDNUMh
cdq
xor edx,edx
div dwSize

mov ebx,pMemory
add ebx,ecx
push ecx

xor eax,eax
add ax,dx
add eax,pMemory

mov dl,byte ptr [eax]
mov cl,byte ptr [ebx]

mov byte ptr [ebx],dl
mov byte ptr [eax],cl
```

Figure 5.6: Reconstructing the file on memory

is possible for a malware to achieve this by adding API into its code which is taken from other applications.

We collected over 300.000 malware samples from Hispasec Company which runs a website to scan samples uploaded by its users.[1] Import Table in PE contains APIs which the program can use upon execution. However, most crypted malware hides its Import Table by calling APIs indirectly. We did not include crypted malware in our tests, however an extension of this work is possible by monitoring API calls of a crypted malware upon execution. After extracting APIs, we tried to classify most used APIs by malware and benign files separately. We selected unpacked unique malware samples which have an Import Table with at least 10 APIs. From the 300.000 samples we processed, we found 72.524 malware which satisfy our conditions. We selected 10.000 samples randomly from

[1] Automated online virus scan page http://www.virustotal.com/

Table 5.1: Detection results for PoisonIvy,Dorkbot, and Boinberg

Antivirus	**PoisonIvy**	**Dorkbot**	**Boinberg**
AVG Free	**Found**	**Found**	**Found**
ArcaVir	Not Found	Not Found	Not Found
Avast 5	**Found**	**Found**	Not Found
Avast	**Found**	**Found**	Not Found
AntiVir (Avira)	**Found**	**Found**	**Found**
BitDefender	Not Found	Not Found	Not Found
VirusBuster Internet Security	**Found**	**Found**	Not Found
Clam Antivirus	**Found**	Not Found	Not Found
COMODO Internet Security	**Found**	**Found**	Not Found
Dr.Web	**Found**	**Found**	**Found**
eTrust-Vet	**Found**	Not Found	Not Found
F-PROT Antivirus	**Found**	**Found**	**Found**
F-Secure Internet Security	**Found**	**Found**	**Found**
G-Data	**Found**	**Found**	**Found**
IKARUS Security	Not Found	**Found**	Not Found
Kaspersky Antivirus	**Found**	**Found**	**Found**
McAfee	**Found**	**Found**	Not Found
MS Security Essentials	**Found**	**Found**	**Found**
ESET NOD32	**Found**	**Found**	**Found**
Norman	**Found**	**Found**	**Found**
Norton Antivirus	Not Found	**Found**	**Found**
Panda Security	Not Found	Not Found	Not Found
A-Squared	**Found**	Not Found	**Found**
Quick Heal Antivirus	**Found**	**Found**	Not Found
Rising Antivirus	**Found**	**Found**	Not Found
Solo Antivirus	Not Found	Not Found	Not Found
Sophos	**Found**	**Found**	**Found**
Trend Micro Internet Security	**Found**	**Found**	Not Found
VBA32 Antivirus	Not Found	Not Found	**Found**
Vexira Antivirus	**Found**	**Found**	Not Found
Webroot Internet Security	**Found**	**Found**	Not Found
Zoner AntiVirus	**Found**	Not Found	Not Found
AhnLab V3 Internet Security	Not Found	Not Found	Not Found
BullGuard	**Found**	Not Found	Not Found

this suitable dataset for our research.

We also processed 10.333 benign files for our test. Benign files are mostly gathered from Windows operating system and softpedia site which contains over 500.000 applications. Most binary samples from softpedia site includes installers which contain same APIs. We excluded installers from our test and used actual programs which are extracted

Table 5.2: Detection results of samples after crypting with our tool

Antivirus	**PoisonIvy**	**Dorkbot**	**Boinberg**
AVG Free	Not Found	Not Found	Not Found
ArcaVir	Not Found	Not Found	Not Found
Avast 5	Not Found	Not Found	Not Found
Avast	Not Found	Not Found	Not Found
AntiVir (Avira)	Not Found	Not Found	Not Found
BitDefender	Not Found	Not Found	Not Found
VirusBuster Internet Security	Not Found	Not Found	Not Found
Clam Antivirus	Not Found	Not Found	Not Found
COMODO Internet Security	Not Found	Not Found	Not Found
Dr.Web	Not Found	Not Found	Not Found
eTrust-Vet	Not Found	Not Found	Not Found
F-PROT Antivirus	Not Found	Not Found	Not Found
F-Secure Internet Security	Not Found	Not Found	Not Found
G-Data	Not Found	Not Found	Not Found
IKARUS Security	Not Found	Not Found	Not Found
Kaspersky Antivirus	Not Found	Not Found	Not Found
McAfee	Not Found	Not Found	Not Found
MS Security Essentials	Not Found	Not Found	Not Found
ESET NOD32	Not Found	Not Found	Not Found
Norman	Not Found	Not Found	Not Found
Norton Antivirus	Not Found	Not Found	Not Found
Panda Security	Not Found	Not Found	Not Found
A-Squared	Not Found	Not Found	Not Found
Quick Heal Antivirus	Not Found	Not Found	Not Found
Rising Antivirus	Not Found	Not Found	Not Found
Solo Antivirus	Not Found	Not Found	Not Found
Sophos	Not Found	Not Found	Not Found
Trend Micro Internet Security	Not Found	Not Found	Not Found
VBA32 Antivirus	Not Found	Not Found	Not Found
Vexira Antivirus	Not Found	Not Found	Not Found
Webroot Internet Security	Not Found	Not Found	Not Found
Zoner AntiVirus	Not Found	Not Found	Not Found
AhnLab V3 Internet Security	Not Found	Not Found	Not Found
BullGuard	Not Found	Not Found	Not Found

from the installers for more accurate result.

We only parsed APIs from ntdll.dll and kernel32.dll in order to make our analysis simpler. Our kernel32.dll includes 1654 APIs and our ntdll.dll includes 2419 APIs. After parsing benign files, we observe that 636 different APIs from kernel32.dll were used more than 50 times in benign files. 110 different APIs are used in benign files from ntdll.dll

more than 50 times. APIs used less than 50 times are not included in our experiments.

Our results show that there are 14 APIs from ntdll.dll which are heavily used in benign files. These APIs are given in Table 5.3. Most of these APIs are functions which require ring-0 privilege. Rootkits most of the time use ring-0 functions in order to communicate with kernel[17] but they are usually in packed format. Packed files are excluded from our experiments. There are 416 APIs from kernel32.dll which do not exist in any malware samples. We think that these results are critical for antiviruses since their detection algorithm relies on detecting malicious and benign behavior. We investigate the methods to copy benign files functionality in next section.

Table 5.3: Most APIs used in benign files which are not common in malware samples

API Name
NtDisplayString
RtlCopyMemory
RtlFindNextForwardRunClear
RtlLookupFunctionEntry
RtlVirtualUnwind
ZwCreateFile
ZwDeleteFile
ZwOpenKey
ZwQueryValueKey
ZwReadFile
ZwSetInformationFile
ZwSetValueKey
ZwWriteFile
__chkstk

5.3 Changing Functionality

We implement an experimental malware which iterates through the import table of the target application and grabs the function name and address. By iterating through the binary of the target application, our malware finds the offset for the call procedure for that specific function. This call procedure is further examined to approximate how many arguments there are and what type of arguments it needs. Function names are stored in the

resource section of our sample malware and they are triggered randomly within 5 sec intervals with appropriate random arguments from its data section. In order to make things easier, instead of adding the function to its own import table, malware uses GetProcAddress to find the exact API address from the function name. We test our malware on 20 different benign applications. As a result, 16 of the resulting malware crashed due to incompatibility of new function's arguments. 4 of them successfully executed new functions that they randomly copied from benign files. These APIs are: ContinueDebugEvent, RegisterServiceCtrlHandlerW, SetUnhandledExceptionFilter, and GetCurrentProcess. Further examination revealed that 3 of these APIs resulted in an error due to invalid arguments. However, these errors did not affect the flow of the program. GetCurrentProcess executed successfully in our experimental malware after grabbing this function from another application.

Our results show that with a better algorithm and randomization, we can produce self-controlled malware which can add or remove its functionality. We used standard genetic algorithm in order to observe how this evolution might work for our malware. We extracted a total of 1242 different benign APIs (population) and formed a deserved state. Our deserved state contains 101 APIs which are used by famous ssh client called Putty [29]. We only used kernel32.dll APIs for this scenario.

We give details of our genetic algorithm parameters as follows:

- Population contains 1242 APIs
- Desired state contains 101 APIs which are extracted from a benign application
- Crossover probability is 90%
- Mutation probability is 0.01%

On an average of 594 generations with 10 consecutive runs, we succeeded in reaching our desired state. Malware combined all APIs of Putty with its malicious functions. We believe that the scoring function of our genetic algorithm can evaluate a benign file by randomly selecting samples from the computer and generating a random desired state instead of using a fixed one. This will help each malware to add new random functionalities

from random benign programs. It should be noted that some APIs are also removed from the malware in order to reach desired state (mutation). We believe that our work is a novel and practical approach to evolution.

5.4 Discussion on possible antiviral solutions against BEMWARE

There are several ways that an antivirus can detect a BEMWARE. Generic prevention techniques for existing propagation methods include detecting a malware based on its fixed code chunk. For instance, antiviruses focus on detecting decryption routine for a polymorphic malware since its encrypted with another key in each generation. The decryption routine is generally fixed and little changes can be done without losing its functionality. Our evolution method which uses standard genetic algorithm is susceptible for such detection.

The genetic algorithm should also be changed in each generation which is a tedious task for virus authors. Antiviral solutions should exploit this and focus on detecting such evolutionary behavior. Fixed code chunks for the evolution algorithm can be included in a signature database in order to detect all variants of a BEMWARE.

Moreover, it is necessary to analyze different samples of the malware to detect its purpose since the evolution algorithm uses a scoring function for each generation. The scoring function clearly represents the behavior of the malware and gives valuable information about its next generation. Behavior based detection features of antiviruses may help to notify user about suspicious activity whenever malware tries to get the list of all functionality that it can produce.

Dynamic analysis methods should also be improved since recent attacks involve emulator's weaknesses and cryptography. Modern antiviruses use whitelist scan technique in order to warn user about a suspicious binary. The whitelist is formed by programs which are trusted by the majority of the population. Engine updates should also use this whitelist to prevent false positives whenever a change occurs in antiviruses' algorithm. This will

quicken the release procedure for an update and helps antivirus to take countermeasures faster.

Chapter 6

Conclusion and Future Work

Malware is advancing more rapidly than ever before. Old methods like polymorphism and metamorphism are widely used but they are not solutions against antiviruses [26][8]. We presented that techniques from the theory of natural selection and behavior models of species can help criminals to develop more efficient malware and we should focus on preventing these types of attacks. We explained different behavior models and presented our ideas about how these behaviors can be used by malware authors. We showed SpyEye and Zbot examples and we expect more virus rivals trying to compete for resources soon. Our research also focused on an evolution algorithm for malware which can change its functionality rather than only signature. The changing functionality of the malware will help us to work on this topic since the power of evolution is limitless.

We expect to see malware which makes use of modern communication channels and social media. Imagine a decentralized botnet communicating through twitter or facebook. It can spread faster with social engineering attacks. This will create a great advantage for the bot since the more machines it spreads to, the higher chance it will have to evolve. Undetectable viruses clearly show that the antiviruses are not a solution to stop malware. However, they are the only option that people can trust for now.

More importantly, we will possibly see more cyberwarfare attacks in the future like the well-armored but unintelligent Stuxnet example. We should predict and work methods for fighting with malware. Electrical power grids, military, power reactors, hospitals,

telecommunication systems, as well as our economy can be controlled with evolutionary malware. They can be used in various disruption and monetary techniques.

One of the great advantages for malware is the speed of technological advancement. Technological advancement also brings new opportunities for malware survival. Enhancing human capabilities by using computational devices is becoming more popular. There are already examples of artificial eyes, hearts, arms or other body parts. If an evolutionary malware spreads into these kind of devices, the problem will be more complex because of the human factor. No one wants an artificial heart with a malware in it, especially when it can think and evolve.

As a future work, we will focus on preventing malware attacks with a rival malware designed specially for this purpose. We succeeded at compromising one of the botnet machines under a DDOS to get the malware responsible for the attack. We examined the malware code and found the server which its gets commands from. With the help of a vulnerability in the server, we were able to shut down the server which the attack originated from. This experience gives us the idea for automating this process and inheriting it in a malware which can be used for good purposes. We believe that the combination of evolution, cryptography, and malware can be the biggest threat or the greatest cure.

Bibliography

[1] C. Adami. *Introduction to artificial life*, volume 1. Telos Pr, 1998.

[2] P.M. Agapow and PM Agapow. Computer viruses: the inevitability of evolution. *Complex systems: from biology to computation*, pages 46–54.

[3] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 31–38. IEEE, 2010.

[4] S.J. Blackmore. *The meme machine*. Oxford University Press, USA, 2000.

[5] D. Bradbury. Digging up the hacking underground. *Infosecurity*, 7(5):14–17, 2010.

[6] T. Brosch and M. Morgenstern. Runtime packers: The hidden problem. *Black Hat USA*, 2006.

[7] D.M. Chess and S.R. White. An undetectable computer virus. In *Proceedings of Virus Bulletin Conference*, volume 5. Citeseer, 2000.

[8] M.R. Chouchane and A. Lakhotia. Using engine signature to detect metamorphic malware. In *Proceedings of the 4th ACM workshop on Recurring malcode*, pages 73–78. ACM, 2006.

[9] F. Cohen. Computer viruses:: Theory and experiments. *Computers & security*, 6(1):22–35, 1987.

[10] F. Corno, E. Sánchez, and G. Squillero. Evolving assembly programs: how games help microprocessor validation. *Evolutionary Computation, IEEE Transactions on*, 9(6):695–706, 2005.

[11] C. Darwin. *The origin of species*. Number 811. Hayes Barton Press, 1958.

[12] R. Dawkins. *The selfish gene*. Oxford University Press, USA, 2006.

[13] AK Dewdney. In the game called core war hostile programs engage in a battle of bits. *Scientific American*, 250(5):15–19, 1984.

[14] A.K. Dixit, S. Skeath, and D.H. Reiley. *Games of strategy*. WW Norton, 2004.

[15] P. Ferrie. Attacks on more virtual machine emulators. *Symantec Technology Exchange*, 2007.

[16] W.D. Hamilton. The evolution of altruistic behavior. *The American Naturalist*, 97(896):354–356, 1963.

[17] G. Hoglund and J. Butler. *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006.

[18] E. Konstantinou. Metamorphic virus: Analysis and detection. Technical report, Technical Report RHUL-MA-2008, 2008.

[19] C.G. Langton. Self-reproduction in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):135–144, 1984.

[20] B. Le Charlier, A. Mounji, M. Swimmer, and V.T. Center. Dynamic detection and classification of computer viruses using general behaviour patterns. In *Proceedings of Fifth International Virus Bulletin Conference*, page 75. Citeseer, 1995.

[21] M.A. Ludwig. *Computer viruses, artificial life, and evolution*, volume 2. Amer Eagle Pubns Inc, 1993.

[22] F. McGillivray and A. Smith. Trust and cooperation through agent-specific punishments. *International Organization*, 54(4):809–824, 2000.

[23] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *Security & Privacy, IEEE*, 1(4):33–39, 2003.

[24] C. Nachenberg. Computer virus-coevolution. *Communications of the ACM*, 50(1):46–51, 1997.

[25] S. Noreen, S. Murtaza, M.Z. Shafiq, and M. Farooq. Evolvable malware. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1569–1576. ACM, 2009.

[26] S. Pearce. Viral polymorphism. *VX Heavens*, 2003.

[27] E.H. Spafford. Computer viruses as artificial life. *Artificial Life*, 1(3):249–265, 1994.

[28] P. Ször. Darwin inside the machines: Malware evolution and the consequences for computer security. 2008.

[29] S. Tatham, O. Dunn, B. Harris, and J. Nevins. Putty: A free telnet/ssh client. *Retrieved July*, 10, 2009.

[30] P.D. Taylor and L.B. Jonker. Evolutionary stable strategies and game dynamics. *Mathematical Biosciences*, 40(1-2):145–156, 1978.

[31] P. Wang, S. Sparks, and C.C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 2–2. USENIX Association, 2007.

[32] A. Young and M. Yung. *Malicious Cryptography: Exposing Cryptovirology*. Wiley, 2004.

[33] H.R. Zeidanloo and A.A. Manaf. Botnet command and control mechanisms. In *2009 Second International Conference on Computer and Electrical Engineering*, pages 564–568. IEEE, 2009.