NTNU
Innovation and Creativity

Detection of Hidden Software Functionality

Jostein Jensen

Master of Science in Communication Technology

Submission date:	June 2007
Supervisor:	Svein Johan Knapskog, ITEM
Co-supervisor:	Maria B. Line, SINTEF IKT

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Software downloaded from the Internet may be very useful, but occasionally unwanted functionality may be hidden in the software, e.g. back doors and Trojan Horses. Malicious code that may turn the computer into a bot possibly as part of a botnet is an interesting case that is yet not fully investigated and documented. A client can be searched for zombie software by undertaking a dynamic analysis of processes, file integrity, network activity and the like. It is necessary to look at mechanisms which are used by malicious code to avoid detection, e.g. techniques used by Root-kits.

Assignment given: 17. January 2007
Supervisor: Svein Johan Knapskog, ITEM

ABSTRACT

Downloading software from unknown sources constitutes a great risk. Studies have described file-sharing networks where the probability of downloading infected files is as high as 70% [1] under certain circumstances.

This work presents theory on malicious software with emphasize on code turning computers into bots and thereby, possibly botnets. It is observed that malware authors start using more advanced techniques to deceive owners of compromised computers. To evade detection, stealth techniques known from rootkits are more and more commonly adapted. Rootkit technology is therefore studied to be able to determine how bots, and other forms malicious software, can be hidden from both automated anti-virus detection mechanisms and human inspections of computers.

The mechanisms used to evade detection by traditional anti-virus tools are in many cases effective. Dynamic behavioural analysis of software during installation is therefore suggested as a strategy to supplement the traditional tools. Several detection strategies are presented, which can be used to determine the behaviour of software during installation. This knowledge is used to design a laboratory environment capable of detecting the mentioned categories of malicious code.

An implementation of the laboratory is provided, and experiments are performed to determine the usefulness of the setup. The software used to set up the laboratory environment are all distributed free of license cost. An evaluation is made and improvements to the system are proposed.

The value of behavioural analysis has been demonstrated, and the functionality of the laboratory environment has proved to extremely useful. Advanced users will find the functionality of the laboratory setup powerful. However, future work has to be done to automate the behavioural detection processes so the public can benefit from this work.

PREFACE

Working with this project has been a challenging task. Malware is evolving at rapid speeds. Research communities do their best in trying to follow and create countermeasures against the threats faced. Trying to find solutions to problems that are not fully solved, neither by research communities nor commercial actors has been inspiring and very instructive.

I would like to thank my supervisor at SINTEF ICT, Maria B. Line. Her input and guidance during this work has been very valuable.

Finally I thank Tuva Gripp, who gave birth to our daughter, Christina Marie, March 12 this year. Parenting is a great responsibility and time consuming task. She has managed to take care of Christina in an excellent way, and at the same time supported me the last couple of weeks during late nights working with this project.

Trondheim,
June 16th, 2007

Jostein Jensen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

API	Application Programming Interface
C&C	Command and Control
DDoS	Distributed Denial of Service
DKOM	Direct Kernel Object Manipulation
DNS	Domain Name System
DoS	Denial of Service
GNU	"GNU is not UNIX"
GPL	General Public License
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IRC	Internet Relay Chat
KVM	Keyboard, Video, Mouse
MAC	Media Access Control
NAT	Network Address Translation
NIDS	Network Intrusion Detection System
OS	Operating System
P2P	Peer-to-peer
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1. INTRODUCTION

Application sharing and distribution on the Internet, where the source is unknown, constitute a risk to the overall Internet security. Often different kinds of malicious code are spread by manipulating applications to contain unwanted, hidden functionality. The software behaviour introduced by ill-intentioned programmers is hard to discover before it is too late. An infected computer may become a relay for e-mail spam, a node in a distributed denial-of-service attack or a stepping-stone for further attacks. These are a few examples, and only the imagination of programmers writing malicious code limit the possibilities.

The threat mentioned above is real. The risk of downloading files with malicious content through file sharing networks is documented by Shin et al [1], who studied the popular peer-to-peer network Kazaa. They found that from the more than 500.000 files they investigated, approximately 15% contained malicious code. When the keyword "ICQ" was used in the search queries, the chances of hitting an infected file was more than 70%. Kalafut et al [2] did a similar study on the Limewire and OpenFT peer-to-peer-networks. Their report shows that 68% of all downloaded executables and archives in the Limewire and only 3% in the OpenFT network were infected.

These numbers speak for themselves. One should not give unconditional trust to anything downloaded from unknown sources on the Internet.

Malware is a term frequently used throughout this report. It is used as a collective term referring to all categories of malicious code in general. The word virus, which is commonly used in the same context, is one of several other categories of malicious code included in the malware term.

1.1. Motivation

The basis of this master's thesis started out as a subject of conversation between the ICT security manager and the security professionals at SINTEF ICT. The topic of discussion was how to determine whether software downloaded from Internet contains hidden malicious functionality or not. Software downloaded from an unknown source with unknown content is every ICT security manager's nightmare.

Botnets and bot code, presented in section 2.2 and 2.3 respectively, are evolutions of traditional malware; from standalone malicious applications to making compromised hosts playing together in ill-intentioned distributed networks. The evolution of malware is a continuously ongoing process, and it is a constant battle between developers of malicious software and anti-virus vendors. It is like a cat and mouse race. Traditionally, detection schemes have been based on signature recognition of known malicious software or activity. However, attackers are constantly improving their code to evade the traditional detection algorithms.

Code obfuscation using e.g. steganography, cryptography, polymorphism and metamorphism[1] are well known techniques, which in many cases effectively prevents anti-virus tools from detecting the malicious content. If the protection shields are successfully circumvented, unwanted ill-natured software can be installed on a computer without the owner's knowledge.

As new signature files for detection are created, distributed and installed, malware previously installed on computers can be discovered and disabled. To prevent this from happening, malware creators may utilize techniques to stay undetected permanently. The very operating system of a computer can be altered to hide the presence of malicious content. Such techniques are known from a malware category called rootkits, further described in section 2.4. Canavan [3] reports that more and more bot code implements rootkit functionality to evade detection.

Sandboxing is a relatively new approach to behavioural malware detection, and the technique has proved to be valuable [4, 5]. The idea is to provide the malicious code with a secure environment where it can be executed as in "real life". An application's behaviour is observed within the sandbox and, based on the behaviour it may be classified as malicious or not. A sandboxed environment does not allow software to communicate with the Internet. However, a trend among malware authors, presented by Symantec [Web1], is to use staged downloads of malware:

[1] Polymorphism and metamorphism are explained in section 2.3.1.2

- Trojan applications are spread on the Internet. These Trojans do not contain any viral code themselves, thus they are not blocked by anti-virus software.
- When these Trojans are run, malicious files are downloaded from the Internet and installed.

Also benign applications like Google Desktop [Web37] use this technique. A small installation package is downloaded from Internet; this application points to Google's repositories where the latest version of the Google Desktop software is contained. Updated software is downloaded and executed when the installer file is run. A modern behavioural analysis environment designed to detect unwanted functionality in downloaded applications, should for these reasons, also allow controlled connections to Internet.

1.2. Outline of this Thesis

With this introduction, it is possible to see the shortcomings of traditional protection mechanisms. This master's thesis will focus on looking at dynamic analysis of software during the installation process, and immediately after the software is installed.

Dynamic analysis is interpreted synonymous with behavioural analysis throughout this report. The idea is to observe software's behaviour to try do determine whether unwanted hidden functionality is present or not.

The main tasks of this project were identified as to:

1. Study literature on the subjects of botnets, rootkits and malicious software in general
2. Based on the literature, develop a method/laboratory setup to be able to detect hidden malicious software functionality with special focus on bot software and rootkits:
    - 2.1. Through dynamic host-based detection – studying activity on the host itself, like observing library calls, running processes, file integrity and more, during and immediately after installation
    - 2.2. Through dynamic network-based detection – studying network related activity during and immediately after software installation
3. Do experiments in the laboratory environment studying examples of bot code and rootkits.

4. Evaluate the value of the methods used based on the experiment results and the literature study.

The intention behind this master's thesis is to develop an analysis environment that can be used in the process of gaining trust in applications downloaded from Internet. Sandbox principles should be used, but with the trends of staged malware downloads, observed by Symantec [Web1], in mind. The analysis results can be used e.g. by ICT managers to make decisions of whether certain software should be allowed or not. The work presented in this report contributes to existing research in the following areas:

1) The dynamic behavioural analysis environment presented in this report is practically free of software cost. The only license cost associated, is connected to the operating system for which the analyzed malware is intended.
2) High-interaction honeypot tools are utilized in a new setting to be able to better control the test environment during tests of e.g. software dependant on an Internet connection to install.
3) The laboratory design is based on descriptions of botnet and rootkit characteristics, combined with updated research on behavioural malware detection principles. Bot functionality is detected during installation, rather than detecting already infected machines through network traffic analysis.

1.3. Related Work

Behavioural malware analysis is previously described by Szor [6] and Skoudis [7]. Virtual machine technology is proposed as a flexible solution for building the laboratory environment. Norman, a Norwegian anti-virus company, has developed a technology called Norman Sandbox [4] which is an automatic behavioural analysis environment based on emulated operating systems and services. Panda Software has released a heuristic scanning engine called Panda TruePrevent [Web17], which uses behavioural analysis to determine the presence of previously unknown malware. Spitzner [8] presents honeypot technology to study hacker behaviour. Research on botnet characteristics are documented, e.g. in [9, 10] and rootkits in [11].

It has proven difficult to find descriptions of behavioural analysis environments solely based on free software. In addition, those environments that are presented are specifically designed as totally isolated systems without any support for controlled connection to Internet. Technologies like Norman Sandbox and Panda TruePrevent represent the direction that the anti-malware industry is moving towards, but to keep their competitive advantage these companies keep their products and technologies proprietary. Most research papers currently published on botnets either describe the functionality of a botnet and its area of application, or botnet detection through network traffic analysis. Honeypot technology is designed and used to study hacker activity from the Internet towards compromised hosts. It is not found documentation of honeypot technology in use to increase the flexibility of behavioural analysis environments.

1.4. Limitations

The work presented in this report is focused towards analysis of malware designed for the MS Windows operating system. It has not been considered whether any aspects of the malware descriptions or detection schemes apply to other operating system.

Due to time constraints and complexity of building one single automated behavioural analysis tool from scratch, existing software will be utilized. Malware detection theory will be presented and analysis tools will be chosen based on properties described in these theories. All software used in the laboratory environment shall be free of software cost. It has not been considered whether any commercial applications are better suited than the utilities presented in this report.

Even though it should be possible to analyze malware, and even benign software, on a general basis in the laboratory environment presented, the focus will be on detecting dynamic behaviour of bot software and rootkits to limit the scope of this work.

A tool called ProcessMonitor, presented in section 3.2.2.4, will be used during the experimentation phase to log every registry-, file- and process event performed on the operating system during the tests. Time constrains will make it difficult to analyze every aspect of the collected log data. Event monitoring is therefore limited to the insertion of registry keys, creation of files and the creation of processes caused by the objects that are to be analyzed. These properties are presented by Szor [6] as especially interesting attributes to study. In addition, it is observed which libraries are loaded into suspicious processes, to be able to say something about what functionality they most probably are designed with.

1.5. Presentation of Previous Work

During the fall 2006, a project assignment was written where techniques for detecting malware through a network-based perspective was explored [12]. The goal of this assignment was to determine whether networking characteristics like local and remote listing of communication ports and observation of network traffic could help finding unwanted malicious functionality during, or immediately after, installation of software. The work led to the design of a laboratory used for a behavioural analysis of software in a network related perspective. The laboratory setup was developed based on three different roles: a victim, an eavesdropper and an attacker.

The victim was the host computer where software to be explored was installed. In addition to the suspicious software, other useful network related analysis tools were installed to help keep track of the network activity. Mainly these tools were used to observe port status before and after installation of the application to be scrutinized.

The eavesdropper was used to collect information about the network activity. A network analyzer collecting every data packet sent on the wire, and an intrusion detection system automatically looking for suspicious activity were used. The victim, the attacker and the eavesdropper roles were installed on separate computers to avoid manipulation of log data by malware controlling the victim.

The attacker was used to do remote, active scanning of the victim. A port scanner and a remote vulnerability scanner were utilized. The port scanner was used to collect port status, which could be compared to results returned by local tools on the victim. The remote vulnerability scanner was used to see if the software installed on the victim had introduced new vulnerabilities to the host. In this process, network activity simulating traffic generated by real life attacker tools was sent to see if the victim responded. During this process known backdoors and other malware could be detected.

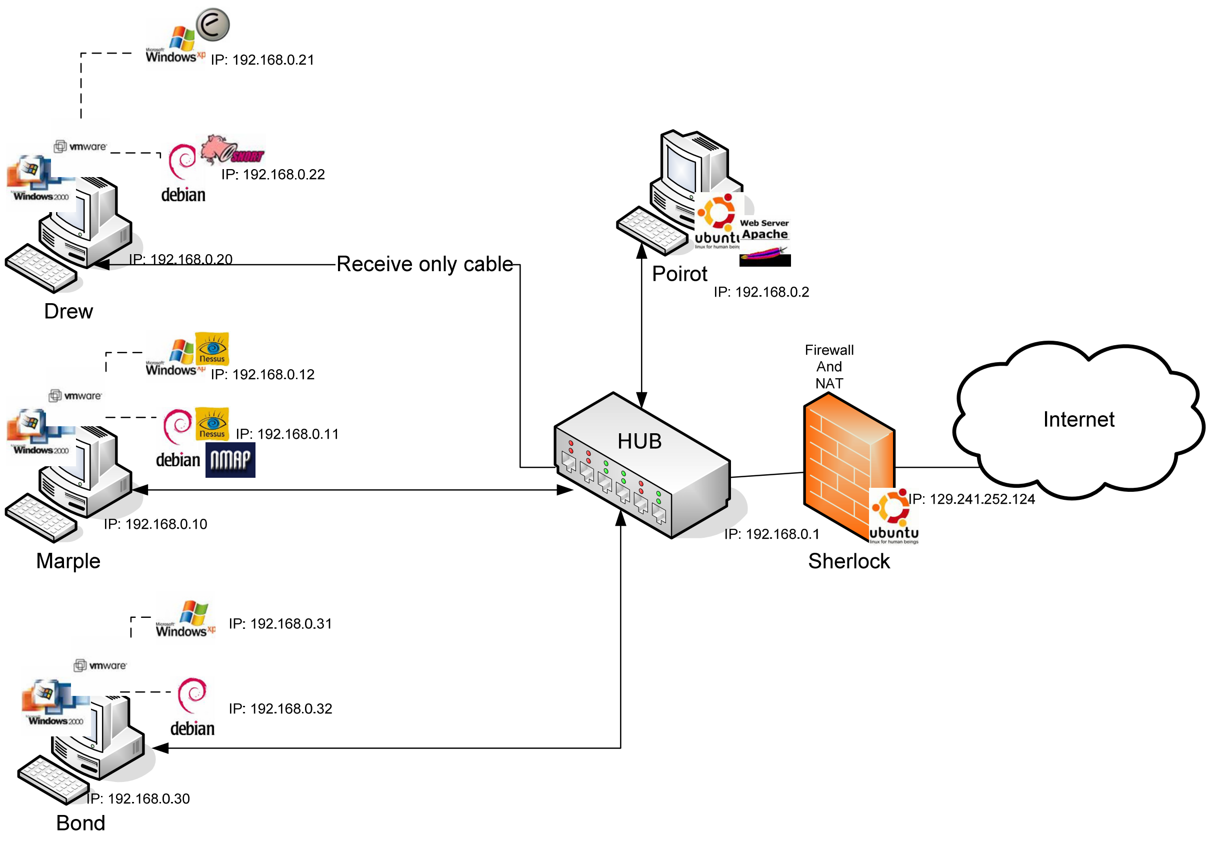


Figure 1: Design of laboratory environment used in the project assignment of fall 2006

Figure 1 illustrates the laboratory environment designed during the project assignment. Bond had the role of the victim; Drew was the eavesdropper and Marple the attacker. The logo images illustrate which operating systems and which tools were installed on the different machines. In addition to the three roles mentioned, a web server, represented by Poirot was introduced to the network. During the test phase it turned out that some malware specimens did not become active before certain web pages was visited. The web server was used to lure the malicious code to activate their networking properties. Sherlock was implemented as a firewall to control the traffic to and from Internet.

In the test phase, malware in general was considered. None of the different malware categories received more attention than others. One IRC based bot, a keylogger and a virus was examined. Google Desktop [Web37] was analyzed as an illustration of a benign application.

The project report concluded that network-based detection can be used to discover certain types of malware. Network worms are typically detected when they start scanning for new victims. Other types of malware, like e.g. those activated by certain types of web page content, are not likely to be discovered solely based on detection through networking characteristics.

The main ideas of network-based detection, presented in the project report of fall 2006 [12], build the foundation for the laboratory environment introduced later in this report. Improvements to the initial setup are introduced in the following work. The main differences are to move away from licensed software, building an environment totally free of software cost, and to make the laboratory setup much more secure from being infected with the malicious code to be analyzed. Further, techniques to observe malware behaviour on the victim host are introduced. Instead of looking at malware in general, the focus of detection will be on bot-code and malicious stealth properties.

Further descriptions of differences and improvements separating the setup presented here and the new laboratory setup can be found in chapter 3.3.

1.6. Research Method

The research process used in this work is separated into four phases, as described in section 1.2. A literature study will help determine which threats are faced, and present current research on how to detect the presence of these. The next phase is to convert the theoretic knowledge acquired into a physical realization of a behavioural analysis environment for software. The third phase is to use both the theoretical and practical skills to analyze software to try to detect unwanted software functionality, and fourth; determine the quality of the environment developed.

In the analysis phase, the process illustrated in Figure 2 is used. A baseline of the test environment is created initially. Each of the analysis tools is used in a data collection phase to determine the state of a known clean system. Next, the application to be scrutinized is installed. A new data collection phase is performed, and the results compared to the baseline. Deviations in the collected results are most likely introduced by the newly installed application. In addition, all file-, registry- and process events generated during installation are recorded. Based on this, it may be possible to determine the presence of unwanted hidden software functionality.

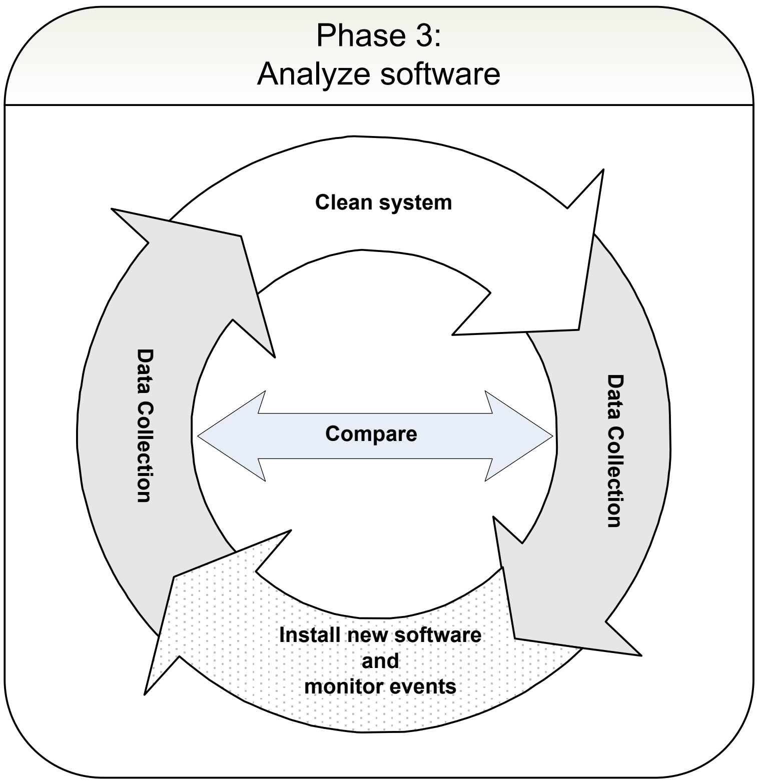


Figure 2: Research process used in the analysis process

To sum up the analysis phase:

1. Update every tool and operating system from reliable sources.
2. Perform data collection on the current, clean system state.
3. Install the application to be analysed and record events.
4. Perform data collection on the new system state.
5. Compare and analyse the results.
6. Revert the virtual machine to a clean state and start over with step 1.

1.7. Structure of the Report

Following is an overview of the different chapters.

Chapter 1: Introduction

This chapter presents some background information and the outline of this thesis.

Chapter 2: Basics

Chapter 2 introduces basic knowledge about malware with a special emphasize on botnet, bot software and rootkits. Behavioral malware detection techniques are also presented. The theory presented in this chapter builds the foundation for the design of the laboratory environment.

Chapter 3: Laboratory Environment

Chapter 3 presents the hardware and software used in the laboratory setup. The laboratory design is presented and argued for.

Chapter 4: Experiments and Results

In chapter 4, the experiments performed and their result are presented. One thorough example is given on the analysis process. The most important findings of the other tests are summarized at the end.

Chapter 5: Discussion

This chapter evaluates the laboratory environment, and future work is presented.

Chapter 6: Conclusion

Chapter 6 concludes the report.

2. Malware

In this chapter an introduction to common categories of malware is presented. Next, botnets, bot software and rootkits are presented in more detail. The information introduced will build the foundation for setting up a laboratory environment and account for the reasoning behind choosing detection techniques presented later in this report; detection techniques used in the process of discovering malicious software functionality.

2.1. Introduction to Malware

Before digging further into the details of botnets and rootkits, a brief introduction to the classic malware categories is appropriate. A basic understanding of the principles behind these concepts will help comprehend the threat faced regarding botnets.

2.1.1.1. Viruses

"A virus is a self-replicating piece of code that attaches itself to other programs and usually requires human interaction to propagate." [7] Viruses need user interaction to be able to replicate. The virus replicates by copying itself to other files or boot sectors of disks once an infected file is opened. The danger of infection is kept within the local host or storage devices physically connected to this. Other computers risk contamination if storage devices containing malicious code are moved between computers or if an ignorant user transmits and opens an infected file on a new computer. Two known viruses are Chernobyl [Web2] and Jerusalem [Web3]. The first viruses were often designed to cause damage on their host computer. In the given examples, Chernobyl was designed to erase the entire hard drive of the infected computer and Jerusalem erased every program run on Friday 13th.

2.1.1.2. Worms

"A worm is a self-replicating piece of code that spreads via networks and usually doesn't require human interaction to propagate." [7] A worm consists of a propagation engine, a target selection algorithm, a scanning engine and a payload. Once a host is infected, a target selection algorithm is used to determine where to look for new vulnerable hosts. According to the chosen target selection, a scanning engine is started to automatically scan the network for new victims. The propagation engine consists of exploits for vulnerabilities. The exploits are used to be able to install the payload onto hosts. This procedure normally continues without

any user interaction. Code Red [Web4] and Nimda [Web5] are examples of two known worms. They were both released in 2001. Code Red was reported to infect about 300.000 hosts in less than 24 hours. Nimda is one of the first worms known to utilize several different infecting vectors.

2.1.1.3.Malicious Mobile Code

"Mobile code is a lightweight program that is downloaded from a remote system and executed locally with minimal or no user intervention." [7] This category includes different kinds of scripts, like javascript, Visual Basic script, ActiveX controls and Java applets. These types of code are normally presented to the user while surfing the Internet. ActiveX controls and Java applets are like small independent software programs. These tools are used to enhance the experience surfing the web in different ways. When visiting a web page with ActiveX or Java applets enabled, the user is normally presented with a choice to download and execute the programs. Like all other software, these controllers may contain hidden and/or malicious content.

2.1.1.4.Backdoors

"A backdoor is a program that allows attackers to bypass normal security controls on a system, gaining access on the attacker's own terms." [7] By installing a backdoor, an attacker is able to gain unauthorized access to a host. Usually administrative privileges are obtained. Full access to the infected host is obtained as long as the backdoor is up and running.

2.1.1.5.Trojan Horses

"A Trojan horse is a program that appears to have some useful or benign purpose, but really masks some hidden malicious functionality." [7] As long as the original intention behind an application is hidden, it can be described as a Trojan horse. ICQ is an example of a benign instant messaging application. A Trojan horse is typically malicious content hidden behind benign looking application names like ICQ. A Trojan horse can contain one or more of the characteristics of the other malware categories. From the introduction to chapter 1, one can see Trojan horses are prevalent and widely used e.g. in file sharing networks like Kazaa.

2.1.1.6.Rootkits

"RootKits are Trojan horse backdoor tools that modify existing operating system software so that an attacker can keep access to and hide on a machine." [7] Rootkits are collections of tools used by attackers. The toolbox may contain backdoors used to obtain unauthorized access to a host at any time, utilities to hide the presence of the attacker and tools to gather information about the system and its environment. A more detailed description of rootkits will be given in section 2.4

2.1.1.7.Combinations

The different malware categories can be combined in different ways. The payload field of worms can be used to distribute viruses, backdoors or rootkits in an automated fashion. As already described, rootkis are tools using Trojan horse and backdoor techniques. Even in the early days of malware, attackers were able to remotely connect to the infected hosts if backdoors were installed. There were, however, no way of using the infected victims as a coordinated, distributed platform. In the next section one can see that properties of the classic malware categories are combined with a remote command and control channel to give attackers access to distributed processing and bandwidth capacity. These networks consisting of compromised computers are referred to as botnets.

2.2. Botnets

Botnets are networks of computers running malicious software, tied together by use of a remote command and control channel. Single computers running botnet software are often referred to as bots, zombies or drones. Rajab et al. [10] define botnets as "networks of infected end-hosts, called bots, that are under the control of a human operator commonly known as a botmaster". Attackers gain control of new hosts using spreading techniques commonly known from other forms of malicious code. Worm spreading techniques are utilized, like scanning and exploitation of vulnerabilities and distribution of e-mails with infected attachments. Distribution via file sharing networks is also common.

2.2.1. Application of use

To learn more about what botnets are, and what the botmasters use them for, the Honeynet Project performed a study on the botnet problem [Web7]. They describe botnets as tools, often used with criminal motives or for destructive purposes. They draw an extensive list of applications of use, among others:

- Distributed Denial-of-Service Attacks (DDoS)
- Spamming
- Sniffing Traffic
- Keylogging
- Spreading new malware
- Installing Advertisement Addons and Browser Helper Objects (BHOs)

During a study, Dagon et al. [13] discovered botnets with sizes ranging from a few hundred victims to one with more than 350.000 compromised hosts. Between July 1 and Desember 31, 2006, Symantec observed more than 6.000.000 distinct bot-infected computers and more than 60.000 of them were active per day [Web1].

Looking at this information in perspective, it is easy to see how botnets can be used to support financial crime. During events like the European soccer championship Euro2004, online betting companies have been contacted by attackers who threaten to take down the betting sites by flooding them with data if a ransom is not paid [Web8]. Bookmakers were demanded to pay £20.000 to £30.000 or face a distributed denial-of-service attack. By controlling botnets of sizes found in [13], the threats are very well accomplishable. Other examples of making money by the help of botnets may e.g. be to collect credit card numbers by collecting data from keyloggers[2] installed on the drones. Credit card information can be extracted from the recorded log files and sold to criminals. Shady advertising companies may also pay botnet owners to target commercials to certain groups of people e.g. by sending spam mail.

Yet another application of use is click fraud. Several advertising companies use strategies for displaying advertisements in similar fashion to the Google AdSense service [Web13]. Site owners are encouraged to display advertisements from Google on their web pages. In return, an amount of money is paid e.g. when the ads are clicked by visitors. Bots like Clickbot.A [Web14] take advantage of these services. To prevent fraud, such as site owners clicking ads

[2] Keyloggers are applications recording every keystroke made on a compromised host.

on their own web pages, the advertising companies use different algorithms to compare number of clicks against IP addresses and so on. Botnets can consist of a great number of computers all with unique IP addresses. Bots specialized on clicking advertisements can be programmed to target the botmasters site. Because several computers with different IP addresses are involved, the fraud is difficult to detect. The financial gain of the botmaster can be considerable.

Botnets have also currently been seen used by politically motivated groups. Around April 27 a massive DDoS attack against Estonian web sites were performed. Access to the web sites of Estonia's prime minister, several banks and schools were effectively blocked for several days. Analysis of the malicious traffic shows that computers from the U.S., Canada, Brazil, Vietnam and others participated in the attack [Web38].

2.2.2. Remote Control

The remote command and control channel is the defining characteristic of botnets. Freiling et al. [14] summarize the usefulness and necessity of a remote control mechanism controlling compromised hosts. Cautious attackers may hard code everything necessary for an attack into a malicious file and distribute this file to a number of computers. It will be difficult to trace this attack back to a certain person. This would, however, be a method without any flexibility, and precise planning in advance of an attack is required. Reverse engineering the code will be an effective way of determining what the intentions were and when the actions would take place. With a real-time remote command and control channel the attacker is able to specify exactly what time an attack shall take place, and which attacks should be performed (DoS, spamming, keylogging etc.). Command and control messages are most often sent in a broadcast like fashion making each bot perform some action at the very same time. In addition, the remote channel may be used to update and upgrade the malware. New functionality can be added and weaknesses in the malicious code can be fixed.

2.3. Bot Software

Barford and Yegenswaran [9] have begun a process of codifying the capabilities of bot malware. In their report, they looked at four IRC based botnet code bases. Each code base was classified along seven key dimensions; botnet control mechanisms, host control mechanisms, propagation mechanisms, exploits, delivery mechanisms, obfuscation and deception mechanisms. In addition, one more mechanism is needed to complete the taxonomy; botnet server localization.

- *Botnet control* refers to the mechanisms used by an attacker to remotely control the bots in his/her network.
- *Host control* refers to the mechanisms used by the bot software to manipulate the victim host once it has been compromised.
- *Propagation* refers to the mechanisms used to search for new hosts to infect.
- *Exploits* refers to the specific methods for attacking known vulnerabilities on a target system.
- *Delivery* refers to mechanisms used to get the malicious code into the host.
- *Obfuscation* refers to mechanisms used to hide the actual content of what is transmitted across a network and what arrives and executes on a host.
- *Deception* refers to mechanisms used to avoid detection once the bot is installed on a host.
- *Botnet server localization* refers to the mechanisms used by bots to determine the address of the botmasters command and control server.

These mechanisms can further be placed in one of two main categories; networking- or host related. To develop appropriate schemes for detecting malicious software functionality, these will represent the main areas to be focused on. Network-based detection of ill intentioned activity is most likely best observed from other computers than the victim itself. Some malware have the ability to manipulate results displayed to a user on a local computer. This category is called rootkits, and is explored in section 2.4. Network related aspects are therefore interesting to observe from a known clean host. Running processes, API calls and so on can, however, only be observed and detected from the victim itself.

In the networking category the botnet control, propagation, exploit, delivery, obfuscation and botnet server localization mechanisms will be studied. The characteristics defined in each of these groups will be deciding when setting up detection procedures for finding network related aspects. Likewise, the host control, obfuscation and deception mechanisms will determine what to look for on the host itself.

Obfuscation is located in both categories because, as mentioned earlier, the term refers to both mechanisms to avoid network-based and host-based detection.

2.3.1. Host Related Properties

The descriptions of host related properties give an introduction to what happens inside the host operating system when installing bot applications.

2.3.1.1. Host Control Mechanisms

A bot is of no use to an attacker if it is not available. Anti-virus software may discover and disable the threat; if not at once, maybe after updating anti-virus signature files. There is also a risk of other malware specimens taking over the system. E.g. observations of two Internet worms competing for the control of a host have been observed [Web9]. Included in the term host control are mechanisms for discovering and disabling anti-virus systems, and eliminating existing vulnerabilities exploitable by other malicious components.

Several other characteristics also fall under the term host control. Bots needs to be activated every time a computer is rebooted. Thus, mechanisms to start automatically are utilized. Entries in files, registry or folders with auto-starting properties are required. These entries are normally inserted at time of installation. Further, bots are designed to perform tasks commanded by the botmaster. Interaction with the host operating system is necessary to be able to perform tasks like key stroke logging, reading information from registry entries or to list and start or stop processes. All of these characteristics have been found when analyzing existing bot source code [9].

2.3.1.2.Obfuscation

Obfuscation refers to techniques used to preserve the program's semantics and functionality while, at the same time, making it more difficult for the analyst to extract and comprehend the program's structures [15]. Obfuscation in a host-based setting are connected to techniques used by malware authors most often used to evade detection by anti-virus software and to make a human analysis more difficult. There are some techniques used by the malware community to achieve obfuscation that are more prevalent than others. These are packing, polymorphism and metamorphism.

Packers are applications originally used to reduce the size of executable applications. Using a packer on a malware specimen results in changes to the malicious file, which may lead to problems for the anti-virus software's signature based detection schemes. The majority of packers combine the compressed executable with the decompression code into a single file. When the packed file is run, the content is decompressed and run from memory. Anti-virus companies provide signatures of known malware packers to help detecting suspicious packed programs.

Polymorphism is an obfuscation technique commonly known from a number of self-replicating worms. The malware is designed to encrypt/decrypt itself with different encryption algorithms and keys every time it is copied or replicated. Some are also able to generate multiple encryption layers. A signature-based detection can only be performed on the decrypted malware body, which will be constant throughout its life [16].

Metamorphism takes the obfuscation one step further. It is a techniques used to change the full code of the malicious application every time it is replicated. Included in metamorphism are register renaming, code permutation, code expansion, code shrinking and dead code insertion [17]. In contrast to polymorphic malware which always is decoded to the same code, metamorphic malware bodies are constantly changing thus making detection extremely difficult. The metamorphic code mutation of the malicious content is performed in such a way that the functionality always remains the same.

2.3.1.3.Deception

Deception mechanisms may be used to avoid detection once the bot is installed on a host. Examples of such mechanisms observed in real life bots are: tests for debuggers, test for VMware, killing anti-virus processes and altering DNS entries of anti-virus software companies to point to localhost [9]. Hiding files and processes are also commonly used techniques [3]. The more professional the botnet owners are the more advanced methods are used. Virtual environments and software debuggers are often used when analyzing malicious code, thus mechanisms to complicate the analysis are implemented by malware authors. Killing anti-virus processes and keeping the malicious running processes hidden are also essential to remain undetected on a computer. The deception techniques used are adapted from rootkits, further discussed in section 2.4.

2.3.2. Network Related Properties

The descriptions of network-based properties give an introduction to how bot applications behave, seen from a network perspective. The activities described in this section are observable from computers remote from the infected hosts.

2.3.2.1.Botnet Control Mechanisms

As mentioned earlier, the defining characteristic of botnets is the inclusion of a remote command and control capability. A bot is controlled by a botmaster via a communication channel. The communication may be realized using different protocols. IRC, HTTP and P2P protocols are commonly used, although botnets using the IRC channels are the most prevalent [9, 10] [Web10].

It should be possible to roughly classify the type of bot by studying network activity. Taking IRC based bots as an example; network traffic containing words like "TOPIC", "PRIVMSG" or "NOTICE" indicate the presence of an IRC based bot [Web7]. These are all messages defined by the IRC standard [18].

In addition to the set of messages defined by the communication standard used, control messages proprietary to the specified bot may be added. Modules and commands for execution of DoS attacks, spamming and so on are not part of any recognized standards. As a consequence these messages will most likely be different depending on the bot developer. Accordingly these messages can be used to categorize which family a bot application belongs to [9].

Automated tools can be used to detect the presence of especially IRC based botnets. As a consequence, some malware authors choose to design their own proprietary communication protocols. Without network administrators knowing what to look for, the risk of being caught is smaller using proprietary vs. standard communication protocols.

2.3.2.2.Delivery, Propagation and Exploit

Rajab et al [10] found that bots use several different techniques to recruit new hosts to the botnet. E-mails with Trojan attachments, mobile malicious code distributed on the web and active scanning to exploit vulnerable services are mechanisms used. Also Trojan applications downloadable from distribution sites or P2P networks [3] are possible delivery mechanisms. Scanning and exploiting vulnerabilities is the most prevalent spreading mechanism.

Bots using scanning to spread can be grouped in two [10]. Section 2.1.1.2 gave an introduction to how worms spread on the Internet. The first group have inherited properties from the worms. Once installed on a host, the bot continuously scan certain ports based on its target selection algorithm and scanning engine. Vulnerable hosts found will be infected. The second group of bots have a variable scanning behaviour. They are equipped with a number of scanning algorithms. The botmaster uses the command and control channel to determine when a scan will start and which algorithm to use.

The scanning engine is normally designed to search for certain vulnerabilities. In the automated bot-code delivery process, exploits for these vulnerabilities are embedded in the bot. It is impossible to know which vulnerabilities are taken advantage of. New exploits are designed whenever new vulnerabilities are found. Bots are often developed with a modular approach, in this case meaning that new exploits can be added whenever a new exploit-module is available. A few of common existing exploits are listed belowe:

- Exploit backdoors left by the Bagle worm on port 2745
- Exploit backdoors left by the MyDoom worm on port 3127
- Exploit Microsoft-DS Service, used for resource sharing on Windows machines, on port 445/TCP
- Exploit NetBIOS Session Service, used for resource sharing on Windows machines, on port 139/TCP
- Exploit Microsoft RPC service, port 135/TCP

2.3.2.3.Obfuscation

Firewalls and intrusion detection systems are utilities constantly looking for suspicious network activity. A bot failing to pass firewall rules will be unable to communicate with the botmaster and is useless to the attacker. Traffic matching IDS signatures will be logged and set off alarms. A vigilant system manager will detect and disable the bot causing alarms. To be able to stay undetected on computers, some bots use different techniques to obfuscate the network traffic.

Encryption, steganography and utilization of unused communication protocol header fields are among the techniques used to obfuscate network traffic. These topics fall under the term covert traffic channels and are further explored in chapter 2.4.2 .

2.3.2.4.Botnet Server Localization

Computers newly infected with bot software somehow need to locate and connect to their botmaster's network of compromised machines. If a bot fails to contact to its belonging command and control server it will not be part of the botnet and thus useless to its botmaster. A white paper from Trend Micro [Web10] describes three different methods used by bots to find their home servers. These are hard-coding of IP addresses, use of dynamic DNS services and use of distributed DNS services.

The use of hard-coded IP addresses and references to static DNS services are in retreat. By shutting down a command and control server which is located by use of static pointers, the whole botnet will be brought down. Bots with static addresses hard coded, will not be able to update and discover new command and control centres. Botmasters using these techniques run a great risk of loosing their entire botnet by the fact that static addresses are easily located and the C&C server shut down.

In stead of using the above mentioned techniques, dynamic DNS services are often used. DNS names offered by dynamic DNS providers, like No-IP.com [Web11] or DynDNS [Web12], are hard-coded into the bot binary. If a command and control server is shut down, a new server can be created and the dynamic DNS entry of the provider can be update with the IP address of the newly established server. When the bot no longer can communicate with their home server, DNS queries are made, and the IP address of the new C&C server is returned.

The third method is for botnets to run their own distributed DNS services. The bot binary contains the IP addresses of these name servers. The address of the C&C server is resolved by contacting the distributed domain name server. These servers are out of the reach of authorities and as a consequence botnets using this mechanism is harder to detect and take down. In addition high port numbers are used to try to evade firewalls and intrusion detection systems.

2.3.3. Bots vs. classic malware

The botnet problem is relatively recent. In the early ages of Internet, most attackers wrote viruses designed to delete files, erase hard drives or perform some other destructive acts. Computers were hacked, often to brag about how many computers were exposed or how difficult it was to break in. There were no means of coordinating the compromised machines using them as distributed computing platforms.

From section 2.2 on botnets and 2.3 describing bot code characteristics, one can see the similarities and differences between bots and other malware specimens. To summarize; bots constitute the evolution of classical malware. From worms the ability to self-replicate across networks are adopted. Backdoors are added to give attackers access to compromised hosts. Trojan horse techniques are used to trick people into downloading bot code from peer-to-peer networks and to make the bot's running processes and registry keys look like benign elements. More recent, bots are also adapting techniques known from rootkits to be able to hide and remain undetected on computers for a longer period of time [3].

The main difference between bots and the other malware categories is the attacker's ability to coordinate compromised hosts via a remote command and control channel. Backdoors also provide access to remote computers via command and control channels. Pure backdoors however, do not allow coordinated operation of several victims at the same time. With bots, each infected host will be part of a larger network, namely botnets, where e.g. one single command directed to the entire network can lead to an instant distributed denial-of-sevice attack. Earlier an attacker would have to log into each backdoor-infected machine sequentially, instructing them to start an attack at a specified date and time, to accomplish the same.

In the next section, rootkit technology is presented. Deception mechanisms to hide the presence of bots become more and more common. In the future, it will be even more important for bot owners to utilize stealth techniques to remain undetected on compromised hosts, as the detection mechanisms become more advanced.

2.4. Rootkit Technology

The term rootkit originates from the Linux environment. The most powerful user with full system privileges on a Linux system is called root. A rootkit is a toolkit running with full privileges on a computer. Section 2.1.1.6 defined rootkits to be a collection of several different tools. Some tools are used by attackers to gain remote unauthorized access to compromised computers. Other tools hide the presence of the attacker and the rootkit files. There are also rootkits containing utilities collecting sensitive information like usernames, passwords, credit card numbers and the like. Rootkits are often used to hide the presence of other malicious files or processes.

Some people argue that rootkits are not inherently malicious [11]. Law enforcement may legally, based on e.g. court orders, use rootkit technology for tapping computers or computer networks. Sony BMG's attempt to prevent piracy of music [Web16] is another example of rootkit technology usage without malicious intentions. However, headlines were created when the public found out, and the "benign" Sony rootkits were exploited by hackers [19]. Even though there are examples of benign use of rootkits, the great majority of users of this technology wear black hats.

The most important keyword describing rootkits is stealth. The whole idea behind rootkits is to provide a practically undetectable playground for attackers on their compromised hosts. Normally, anomalies in lists of running processes, opened communication ports, new unknown registry settings and the like, should make a vigilant system administrator do further investigations. The most common way for attackers to prevent detection, besides rootkits, is to use Trojan techniques giving processes, registry keys and so on benign looking names. These elements are nevertheless detectable, and rootkis are the attackers' response to administrators on the alert. By manipulating responses returned by applications or operating system kernels, attackers can prevent certain information from ever being presented to the user. There are two main categories of rootkits: user-mode and kernel-mode.

2.4.1. Rootkits – Host Manipulation

The following section describes the main principles used by rootkit authors to control compromised computers and to provide stealth. The descriptions refer to techniques utilized on the Microsoft Windows platform. It is not considered whether the same techniques apply to other operating systems.

2.4.1.1.User-Mode Rootkits

User-mode rootkits are the more primitive of the two categories. The first user-mode rootkits were Trojan versions of known applications. Netstat is a tool displaying open communication ports on a system. A Trojan version of this application could prevent open ports belonging to the rootkit from being displayed. Likewise, file-listing applications can be configured not to display files belonging to or hidden by the rootkit.

Another user-mode approach is to replace existing libraries with wrappers. Wrappers contain the same API functions as the original libraries, but also include filters set by the attacker. Hoglund and Butler [11] use the term hooking for this functionality. Each library contains tables of how each API function it offers can be contacted by external libraries en executables. Rootkits identify these tables and modify them to point API function calls to their own malicious libraries. When an application performs an API call, the request is directed to the Trojan library which redirects it to the original library. Necessary system calls are made and data processed. The Trojan library filters data returned by the original library and modified information is displayed to the user [20]. Figure 3 illustrates this process.

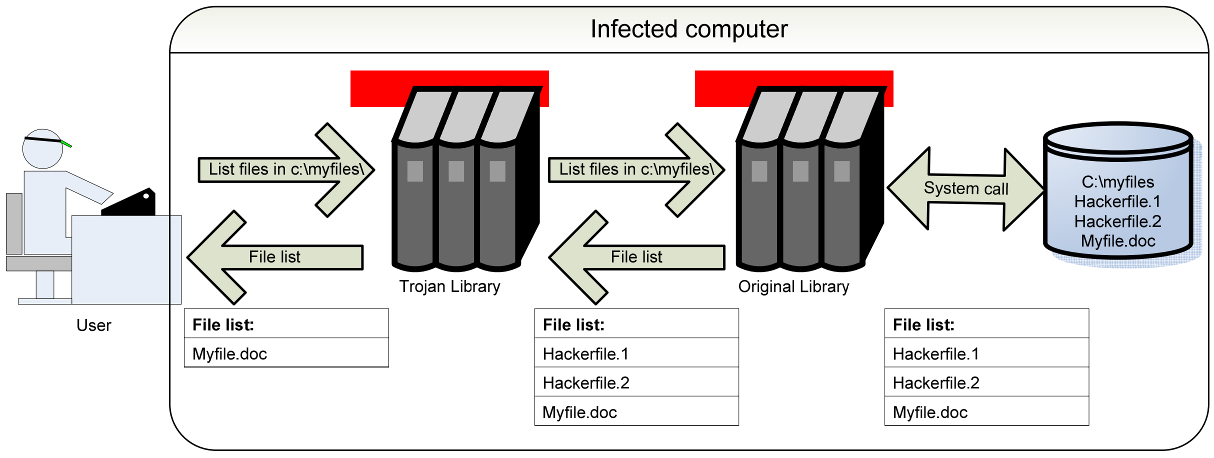


Figure 3: Illustration of user-mode rootkit operation. Illustration adapted from [20].

The first Trojan user-mode rootkits have one severe weakness. To be able to hide files, folders, processes, registry keys and every other aspect manipulated by the malware, all applications able to report anything from these elements have to be controlled. In the netstat example, where open ports are listed, one would also need to control third party tools like the popular port listing utilities fport [Web49] and TCPView [Web50] to be able to hide perfectly. Compromised systems running Trojan versions of netstat and third party tools not controlled by the attacker can easily be discovered by comparing results from the different applications. A result displayed in one application, but not in another, should call a system manager's attention.

Using Trojan libraries improves this situation. Attackers will need to identify which library calls popular applications make, and insert Trojan libraries handling these calls. Several applications probably use the same techniques to retrieve information from the operating system, thus it is easier to control a limited amount of libraries than every existing application. One can hardly be certain every possible API-call is covered. Comparing results from applications using different techniques to retrieve information could be a source of discovering user-mode rootkits using both application manipulation and library manipulation.

2.4.1.2. Kernel-Mode Rootkits

The second category of rootkits is called kernel-mode rootkits. These are the underground community's response to user-mode rootkits' weaknesses. By manipulating the operating system kernel one can control close to every aspect of a computer.

In order to allow flexibility, MS Windows is designed with a layered driver architecture. Hardware devices in a computer are dependant on a set of drivers to function properly together. Whenever access to a device is requested, the request is passed down a stack of drivers. This allows each driver to handle a limited number of tasks. To be able to add and remove functionality at a later stage, Windows drivers are built with a modular approach. New drivers can be added to support new functionality.

Several drivers are involved e.g. when displaying information to the screen. A user requests information by typing on his keyboard. The keyboard driver is called and passes the request to the next driver in the chain, which might be the file system driver. If the requested information on the file system is encrypted, an encryption driver might be called before passing the information on to the display driver. Finally, the wanted information is displayed on the screen.

Regular software is not allowed to communicate directly with the operating system kernel. Access to the kernel is gained by rootkit authors by writing drivers accessible by user applications. A driver typically operates in the kernel space of the OS. Because of the modularity of the driver structure, attackers can utilize malicious drivers to intercept regular data flows in the kernel. By hooking an ill-intentioned driver into the regular data flow between drivers, an attacker can by far manipulate every operation made on a computer.

The insertion of a malicious driver can be utilized in much the same way as a user-mode Trojan library. The ill-natured driver can reprogram function tables in the kernel in such a way that every interesting event is processed by the rootkit code. Controlling the information flow to and from the kernel means full control of the computer. Keyloggers can be made from drivers hooked into the keyboard driver; files can be hidden by programming filters in drivers hooked into the file system driver and so on. Hoglund and Butler [11] also mention the possibility of creating audio bugs, webcam controllers and the like by utilizing these techniques.

A second method, besides the driver insertion, used to create kernel-mode rootkits is first presented by Butler [11] and called Direct Kernel Object Manipulation (DKOM). Information on running processes, loaded device drivers and open ports is accounted for in structured objects running in memory. Objects running in memory can be altered directly. It is not necessary to hook API calls and filter results, which was the case for user-mode rootkits and kernel-mode rootkits using driver insertion.

A structure called EPROCESS contains doubly linked lists of all active processes currently running in the Windows operating system. Each object in the structure contains links to the objects in front of, and behind itself. Windows Task Manager, among other tools, uses this information when listing all running processes. The object structure is traversed by following the pointers to the next object. To hide processes, the DKOM technique takes advantage of this and manipulates objects in the EPROCESS structure. Object links pointing to rootkit processes can be altered to ignore the malicious process and point to the next benign process. Figure 4 illustrates this situation.

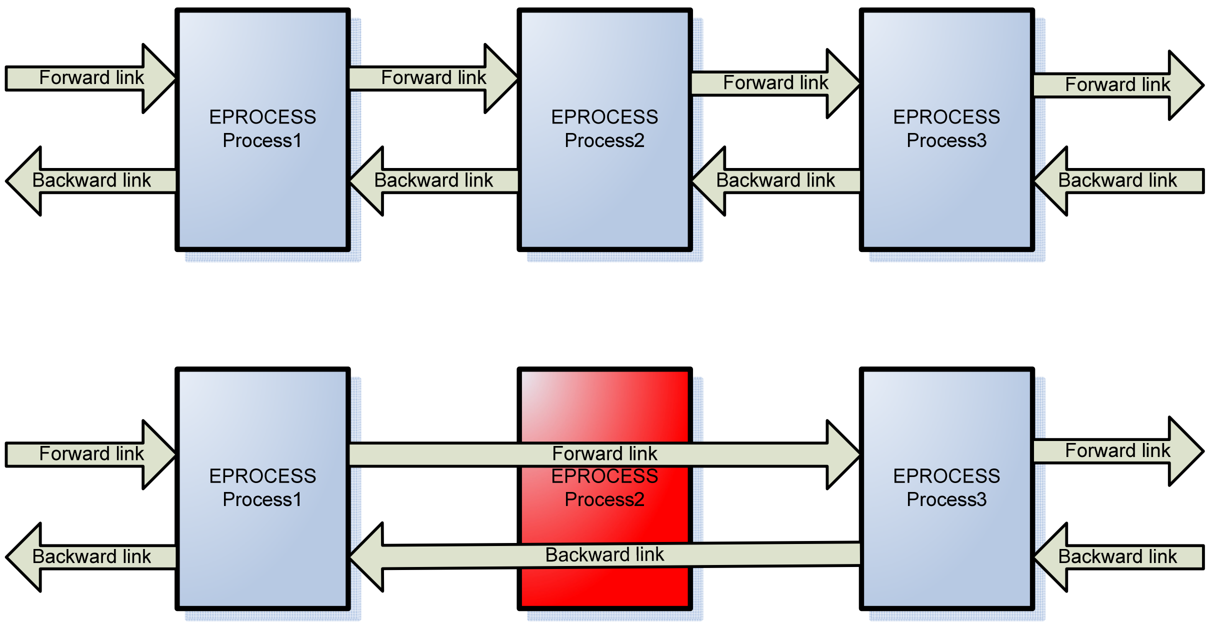


Figure 4: Doubly linked structures keep track of running processes. The red process is un-linked from the structure using DKOM techniques.

The topmost illustration represents the doubly linked list used by Windows to keep track of running processes. The illustration below shows the same list manipulated to hide a rootkit process using DKOM techniques.

Information of every running process, including the hidden once, is kept elsewhere in the operating system kernel. Unlinking objects from the EPROCESS structure does not affect the CPU time acquired by the malicious process.

In similar ways as hiding processes, DKOM can be used for hiding loaded device drivers and open ports. These elements are stored in structures like EPROCESS, and can be manipulated using the same principles.

In addition to contain information about neighbour elements, other useful information is stored using the mentioned data structures. Among other attributes, information of which privileges a process runs with, ant the like, is stored. An attacker in position to manipulate pointers to other elements is also capable of elevating system privileges using DKOM.

While user-mode rootkits could be discovered by comparing results from several utilities using different techniques in retrieving information, kernel-mode rootkits control the system on such a low level that the same detection techniques will most probably not apply. Rootkits control the data flow to and from the kernel in such a way that the integrity of any data, even analysis results from malware detection tools, can not be trusted. In the end, who wins the battle is a matter of who controls the kernel first.

2.4.2. Rootkits – Covert Traffic Channels

Clever attackers do everything they can to remain undetected on a system. Botnets, for instance, are dependant on the command and control channel. Network activity related to communication between a zombie and a botnet server is vulnerable to detection. Network monitors like IDS solutions are often used both on corporate networks and personal computers, and administrators guarding their systems can easily notice anomalies in network activity. To remain undetected on a network, stealth technology can also be applied to the communication channels. Regarding botnets, it is hard to imagine that a DoS attack can be performed undetected from a supervised system due to the amounts of data being transmitted at time of the attack. Remote control commands, transfer of sensitive data like password files, logs containing keystrokes and other user activities are, however, very well possible to transmit using stealthy approaches.

Hoglund and Butler [11] cover some topics of covert channels in their book on rootkits. The key aspect for those designing covert channels is to implement a communication path that breaks through firewalls undetected by intrusion detection systems or other network sniffers. These channels are often created using some form of extensions to existing communication protocols and are specifically designed to move hidden data.

If a covert channel is detected and monitored, it is unlikely to find any clear text data. Encryption and steganography are techniques used to obfuscate data being transmitted. In [11] an illustration of how DNS queries can be used to generate covert channels is given. The illustration is copied in Figure 5. The DNS service is essential in the operation of most networks, hence DNS traffic is normally allowed through firewalls. In addition, the DNS protocol uses UDP packets which minimize traffic overhead. TCP uses a three way handshake to establish connections, while UDP does not. It is also possible to spoof the source address of UDP packets.

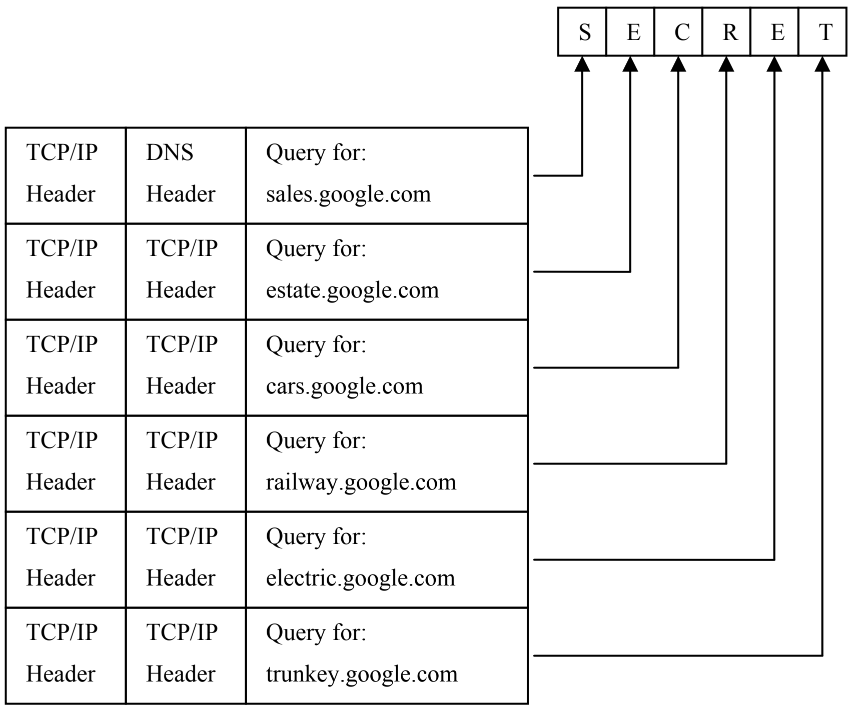

S
E
C
R
E
T
TCP/IP Header
DNS Header
Query for: sales.google.com
TCP/IP Header
TCP/IP Header
Query for: estate.google.com
TCP/IP Header
TCP/IP Header
Query for: cars.google.com
TCP/IP Header
TCP/IP Header
Query for: railway.google.com
TCP/IP Header
TCP/IP Header
Query for: electric.google.com
TCP/IP Header
TCP/IP Header
Query for: trunkey.google.com

Figure 5: A series of DNS requests used to encode an acrostic message. The first letter of each DNS name is used to reconstruct the message "SECRET" [11].

Figure 5 illustrates how steganographic principles are used in combination with DNS queries to create a communication channel transmitting data in a hidden fashion. Using the principles shown in the figure above, several key properties of a covert channel can be accomplished:

- Firewall evasion is made by using the DNS protocol (UDP port 53) for outbound connections.
- The transmitted message is obfuscated using steganography, thus more difficult to detect.
- A random time interval can be used between each packet, avoiding a peak in network activity.

In addition to the steganographic principle used above, a second layer of protection can be added. By encrypting the message before it is transmitted it will be even more difficult to detect the communication channel. An illustration of this can be given by use of the simple Caesar cipher:

Plaintext:	A B **C** D **E** F G H I J K L M N O P Q **R** **S** **T** U V W X Y Z
Cipher text:	D E **F** G **H** I J K L M N O P Q R S T **U** **V** **W** X Y Z A B C

The key in this example cipher is 3, meaning the alphabet is offset three letters to the left between the plaintext and the cipher text. The encryption is done by substituting a plaintext letter with the corresponding cipher text letter.

"SECRET" would be encrypted to "VHFUHW". A vigilant system manager being able to discover a system in the executive messages in the first example would have more trouble discovering relations between the following DNS queries:

- **v**irginia.website.com
- **h**ouston.website.com
- **f**lorida.website.com
- **u**tah.website.com
- **h**awaii.website.com
- **w**ashington.website.com

The examples given so far are illustrations of covert channels implemented in payload fields of data packets. There are several examples of hidden channels carrying data in protocol header fields. HTTP headers [21], TCP headers [22, 23], IP headers [23] and ICMP packets [Web15], among others, can all be carriers of malicious network traffic. Attackers take advantage header fields with optional use or fields not used at all to remain undetected. These are normally not inspected by firewalls or IDSs, thus these packets pass unnoticed through the network.

There are even descriptions of covert traffic channels utilizing lower layer protocols [24] and timing between received packets [25] to establish communication. These channels do not send any bits or bytes in the data packet itself. Rather sophisticated methods are implemented to influence and interpret the behaviour of e.g. packet collisions in the MAC protocol or timing and order of received IP packets.

2.5. Malware Detection

So far characteristics of malware are presented with a special focus on botnets and rootkits. This section will introduce different theories, which can be used to determine how and where to look for malicious content in general, through behavioural analysis. The section is summed up by looking at how each detection theory can be used to find properties specific to bots and rootkits.

Most traditional anti-virus systems use signature-based detection, meaning that code introduced to the computer is scanned for characteristic byte sequences. Malware instances can be recognized by looking for unique patterns in the code. This method proved to be effective in the beginning of the battle between malware writers and anti-virus vendors. The major drawback to this technique is the absence of ability to detect new, unknown malicious code. When new instances[3] of malicious software are found, new signature files have to be written and distributed to the detection tools. As a consequence there is always a window of vulnerability between the release of a new ill-natured code sequence and the release and full update of signature databases.

As traditional malware detection techniques has become better and better, virus writers has twisted their minds to come up with new and improved ways for their code to escape detection. The signature-based detection techniques proved to be effective in discovering known malware specimens. To extend the lifetime of their code, malware writers started using polymorphic and metamorphic techniques. Each time the malicious code is replicated a mutation of its code takes place. As a consequence, a new signature-file has to be written for

[3] A new instance can either be a mutation of an existing malware specimen, a new malware specimen or malware utilizing zero-day attacks.

each mutation to be able to detect the mutated specimen. With close to infinite permutations, ill-natured code can prolong its active life considerably.

Limitations in the signature-based detection techniques force the malware fighters to implement new and improved methods in the combat. Behavioural analysis is proposed as a complement to signature-based detection techniques. The following sections will introduce some of the research made on the topics of behavioural malware analysis.

2.5.1. Host-Based Detection

This section presents techniques that can be utilized to observe changes in the host operating system, during or immediately after installation of an application.

2.5.1.1.Auto-Start Extensibility Points

To complement signature-based tools in the malware detection process, Wang et al [26] suggest complementing the traditional approaches with a concept they refer to as Auto-Start Extensibility Points. In [26] a focus on the spyware category is presented, but the techniques introduced will apply to several other malicious software categories as well. Wang et al state that: "an overwhelming majority of all spyware programs infect a system in such a way that they are automatically started upon reboot and the launch of most commonly used applications".

Auto-Start Extensibility Points are application extensibility points offered by the operating system, which can be utilized by applications to ensure they are automatically started after every reboot of a computer. The ability to start automatically after a reboot is crucial to close to all malware categories. It would be almost impossible to build botnets if the code did not survive a reboot. Worms would be far less efficient, and why bother to establish a backdoor if it was not possible to reconnect to a compromised computer the next day? Software using the auto-start extensibility points are distinguished by two categories [26]:

1. Standalone applications that is automatically run by registering as an OS auto-start extension, such as a Windows NT service

2. Extensions to existing applications that is automatically run, or extensions to popular applications commonly run by users, like web browsers

The solution presented by Wang et al is to monitor the auto-start extensibility points to determine if unauthorized code is run during start-up. Typical places to look for auto-starting code is in registry settings where certain registry keys are dedicated to auto-starting applications and where e.g. Internet Explorer toolbars are registered. Also start-up folders and drivers have to be monitored.

The technique proved to be efficient in discovering spyware, but the method did have some limitations. Among others, rootkits hide their presence in such a way that other detection techniques are necessary, and the malware can use undocumented methods to start automatically.

2.5.1.2.Cross-View Diff

Wang et al [27] introduce a concept called *cross-view diff* to detect stealth software. As described in section 2.4, user-mode rootkits utilize techniques to intercept OS-provided APIs. Queries initialized by the user are filtered to hide certain files, registry settings or running processes. The idea behind the *cross-view diff* approach is, for instance, that queries listing every file on a hard-drive can be made on different levels.

A high-level approach will typically use the OS-provided API to list the points of interest. These results can be compared to a low-level scan of the computer. A low-level scan will typically be initialized from within the OS-kernel e.g. by use of drivers. While user-mode rootkits most probably will be able to manipulate results from the high-level scan, they are not likely able to affect the low-level scan results, thus deviations in the scan results indicate some suspicious presence.

The authors of [27] argue that *cross-view diff* can be used to detect hidden files, hidden registry keys and hidden processes. There are however some limitations. As mentioned, the method is only fit when trying to detect malware utilizing hiding techniques similar to ones found in user-mode rootkits. As explained in section 2.4.1.2 kernel-mode rootkits operate inside the kernel. One can never trust results returned from a kernel already infected with this type of malware. Since the malicious code is able to affect both the low- and high-level scans, a positive detection can prove difficult.

2.5.1.3.Cross-Time Diff

Cross-time diff is also a term introduced in [27] (2004) although the concept is presented earlier e.g. by Kim and Spafford in [28] (1994). Cross-time diff aims at comparing the state of a running system with a previous snapshot of the same system. Essentially, snapshots from two different points in time are compared as opposed to cross-view diff, which compares two different points of view at the same time.

File integrity checkers like Tripwire [28] are examples of cross-time diff applications. These tools came as a response to the first user-mode rootkits, which consisted of Trojan versions of known Unix applications like ps[4], ls[5] and netstat[6]. An integrity checker can scan a computer for files, registry settings and the like and store a snapshot of the current state in the form of message digests of each file and setting. At a later time, the integrity checker can be run once again. Changes in known operating system files or important registry settings can indicate the presence of Trojan horses or other malicious code. By doing a complete integrity check of an entire hard-drive, the tool can be used to determine which files were installed and other changes made by a newly introduced application.

Szor [6] illustrate some problems using integrity checkers alone to detect malicious components. False positives can be created by applications changing their own code. Configuration information is sometimes stored in the executable itself, thus the integrity checksums may differ for each scan. Packers can be used to save disc space and security updates may introduce new operating system files and changes to existing once. The major problem with integrity checkers as stand-alone malware scanners though, is the lack of ability to detect malicious code running in memory and the ability to detect malicious objects hidden with rootkit technology.

[4] ps is a utility listing all running processes

[5] ls is used to list the files in an archive

[6] Netstat is used to display information like open communication port and network statistics

Integrity scanners are dependant of objects to analyze. The first user-mode rootkits could be detected as they introduced changes to existing operating system files as mentioned above. With new generations of rootkits, the existing operating system files are intact. Rootkits manipulate the information flow between users and objects. A file request from an integrity checker will be forwarded by the rootkit code to the original files. Since the original files are intact, they will be declared clean. *Cross-time diff* techniques will always suffer from these drawbacks. However, combining integrity checking with other detection methods will increase the chances of detecting malicious activity [6].

2.5.1.4. Event Monitoring

Moffie et al [29] suggest complementing traditional signature-based malware detection techniques with an analysis of data flow and control flow information collected during the execution of a binary. There are numerous events that need to be monitored. In [29] looking at system calls, library calls and properties like file and process creation is proposed. System call traces may also be used by intrusion detection systems to monitor malicious activity [30].

System calls are the mechanism used to pass information between an application and the operating system. In other words, to communicate with the OS kernel, applications utilize system calls. Read and write operations to the file system and initiation of network connections are examples of functions depending on an information flow between applications and the kernel. Library calls are the mechanisms used by applications to request services from other applications or libraries.

To be able to detect malicious processes resident in memory, the memory needs to be scanned [6]. Some malware only run as processes in memory and do not interact with the file system. Such processes can be detected either in user-mode or kernel-mode. The same problem is faced here as in other detection techniques. Rootkit technology can be used to hide processes. Kernel-mode scanning may help detect user-mode rootkits. Kernel-mode rootkits, however, may even manipulate kernel-mode scanning results and thus prove difficult to detect.

Monitoring the above mentioned properties will give detailed information during the installation process. Among other aspects, it will be possible to determine which files were created, which processes were started and where and what information was collected by the malware during installation. It will also be possible to determine if network connections was established. One can be able to decide whether or not certain behaviour belongs to malicious processes based on the collected data.

2.5.1.5. Dead or live analysis techniques

Carrier [20] describes the problem of live vs. dead analysis of digital systems. A live analysis uses software running on the compromised operating system during the analysis process. A dead analysis, however, refer to an analysis of a system while the compromised operating system is powered off. A trusted OS and trusted applications are used to investigate the content of the target disk drive.

Remembering how rootkits interfere with library calls, system calls and even data flows inside the kernel, it is easy to see the limitations of live analysis techniques. There is always a risk of data being manipulated and filtered as long as the rootkit is active. During a dead analysis the infected operating system is inactive, thus the elements hiding the malicious content is also inactive.

Even though these thoughts seem promising in detecting rootkits, they will not be further explored in this report. They are mentioned here to illustrate that there are other discovery methods than the ones presented above.

2.5.2. Network-Based Detection

Network-based detection refers to techniques used to discover the presence of malicious entities by studying properties related to network activity. As presented in section 1.5, network-based detection of malicious activity was studied during the project assignment of fall 2006 [12]. To complete this report, some of the topics presented in [12] are repeated. The topics of access control records and honeypots, below, were not included in the fall assignment.

2.5.2.1. Access Control Records

Router Access List is proposed by Szor [6], among others, as a defensive strategy used to protect networks. The idea is to update routers with records of hosts or network segments of which are allowed or denied access to network resources. Data packets from intruders may simply not be routed in the network because of the policies defined in the access lists.

Briefly mentioned in section 2.4.2 the source address of UDP packets can be spoofed. Address spoofing is e.g. used by some malware specimens to complicate the task of determining which host is infected on a network. In a behavioural analysis scenario, access control records can be utilized to help detect usage of address spoofing. Network traffic from hosts that should not send any traffic or from addresses not allocated to any hosts, can be caught and logged based on the access control records.

2.5.2.2. Intrusion Detection Systems

Intrusion detection systems (IDS) are utilities used to inspect both network flow and the content of each packet sent on a network. Network intrusion detection systems, NIDS, are designed to sniff packets off the wire and based on certain rules, decide whether the network traffic is malicious or not. This can be done either via a signature-based approach or an anomaly-based approach.

The signature-based approach can be compared to traditional anti-virus detection mechanisms. Network activity is compared to a list of predefined signatures based on unauthorized communication protocols, known attack patterns, known malicious payload content and the like. Anomaly-based detection will try to determine the presence of malicious processes e.g. by studying the amount of network traffic compared to time of day, or abnormal traffic flow patterns. The assumption is that the network behaviour of an attacker or malicious process is noticeably different from that of a legitimate user.

Mukherjee et al [31] suggest using IDSs to detect the presence of malware or intruders on a network. Backdoors can be detected by discovering traffic flows to and from e.g. unauthorized communication ports, and payload fields can be scanned for messages characteristic to e.g. IRC bots [10] [Web7]. Scanning patterns and replication of worms are also possible to detect. Polymorphic worms have proved to make IDS-based detection more difficult, but recent research has demonstrated that even such obfuscation techniques are possible to handle [32].

Intrusion detection systems are great tools for detecting unwanted traffic on a network. They may, however, be difficult to tune in such a way that administrators are not flooded with false alerts on malicious activity or even more problematic, not alerted of serious incidents at all. This is the major drawback to these systems.

2.5.2.3. Honeypots

Spitzner is one of the leaders of the honeypot development, and a lot of his work and concepts of honeypots are published in his book [8]. Honeypot theory can also be studied in [33]. Spitzner's definition of a honeypot is: *"A honeypot is security resource whose value lies in being probed, attacked, or compromised."* A honeypot is a system intentionally left with minimal inbound security and a more firmly defined outbound policy. By letting attackers, either manually or automatically (worm propagation), gain access to honeypot systems, one can observe the activity and learn more about current Internet threats. Honeypots have no production value, and no person or resource should be communicating with them. As a consequence one can argue that most traffic to and from the system is a result of malicious activity.

Honeypots can be used in production environments to help raising alerts of unauthorized activity on the network and to slow down worms. They can also be used as research resources to study present threats. Honeypots are categorized in three different interaction levels; low-interaction, medium-interaction and high-interaction.

Low-interaction honeypots typically emulate a limited set of services, e.g. telnet or ftp. As there are no real services to log on to, connection attempts will be rejected each time. Each attempt to connect to the honeypot is logged, and will give valuable information like which services are tried exploited, which hosts initiated the attack and the like.

Medium-interaction honeypots offer more fully implemented services. Instead of rejecting each connection attempt, a medium-interaction telnet service, for instance, could allow the attacker to log in, list and copy files and so on. The service is still emulated so the attacker is not able to interact with the real operating system. Such systems will return more information about malicious activity than the low-interaction system, but at a price of more complexity.

High-interaction honeypots give attackers access to real operating systems. The attacker is free to play around on the compromised system. By studying the activity one can learn how attackers break into systems, what they look for on compromised hosts, which utilities they install and the like. If not properly controlled, a high-interaction honeypot can even be an active player in attacks towards other systems. To prevent this, an interesting set of tools has been developed.

Honeypot concepts are interesting and may also prove to be useful in behavioural analysis of malicious applications.

2.5.2.4.Remote Vulnerability Checking

Malware enabled to communicate with its environment utilizes different techniques to establish the communication. Some ill-natured code initiates every connection from within the host, while others establish listening ports waiting for an attacker to establish the session. Skaggs et al [34] suggests using remote vulnerability scanners to detect the presence of malware on a host.

Remote port scanners can be used to determine which ports are set in listening mode on a computer. One technique used by attackers to establish connections to backdoors on a system is to activate a dedicated listening port. A remote port scanner will be able to detect active ports on a host. Abnormal port status may indicate the presence of unwanted functionality. While a port scanner is able to detect open ports, a vulnerability scanner performs several checks on opened ports to fingerprint the services running [34]. Various sequences of data can be sent to each port. The responses can be analyzed and compared to a list of signatures. This can help the user to evaluate which ports belong to legal services and which are opened by unauthorized software.

Listing ports from a remote location serves at least one important advantage over listing ports locally on a host. Stealth software is able to manipulate the port status displayed to a user locally, but not the results collected remotely. Vulnerability scanners can add valuable information to an analysis, but suffer from the same drawbacks as other signature-based tools.

2.5.3. Botnet and Rootkit Detection

This section will try to combine the theory on botnet and rootkit behaviour with the theory presented on malware detection. This will give guidance to how a behavioural analysis environment should be designed logically and physically, and what functionality is needed to detect the malicious instances.

Looking at the host- and network related properties of bot software one by one; it is possible to determine which detection techniques should be best fit to discover the unwanted functionality during installation. This section sums up the theory presented on bot, rootkits and malware detection.

2.5.3.1.Bot Detection – Host-Based

Host control, presented in section 2.3.1.1, describes the mechanisms used by the bot software to manipulate the victim host once it has been compromised. Among other properties, adding functionality to automatically start after a computer reboot and patching of existing vulnerabilities are included. Wang et al [26] suggested monitoring auto-start extensibility points to determine installation of unwanted functionality, as presented in section 2.5.1.1. Hence, the registry or file entries a bot uses to start after a reboot should be able to detect through these principles. A remote vulnerability analysis before and after installation, can help determine if certain vulnerabilities are patched. Remote vulnerability checking is described by Skaggs et al [34], and presented in section 2.5.2.4.

Obfuscation, presented in section 2.3.1.2, refers to mechanisms used to hide the actual content of what arrives and executes on a host. Introduced in section 2.5.1.4, Moffie et al [29] describe event monitoring as a method to determine what is really installed onto the computer. Obfuscation is a trick used to avoid signature based detection, thus during or after installation the functionality is the same each time the malware is run. File activity, process creation and the like can be observed to determine if some unwanted activity is introduced.

Deception, presented in section 2.3.1.3, is described as mechanisms used to avoid detection once the bot is installed on a host. Deception mechanisms are strongly related to rootkit technology. Detection of bots utilizing deception strategies is therefore connected to rootkit detection, which is presented in section 2.5.3.3

2.5.3.2.Bot Detection – Network-Based

Botnet control mechanisms were presented in section 2.3.2.1, and refer to mechanisms used by an attacker to remotely control the bots in his/her network. The most prevalent communication channels used by botnets are built on the IRC protocol. Also peer-to-peer protocols and web-protocols are used. Intrusion detection systems, presented in section 2.5.2.2, can be configured to automatically detect the communication channels based on protocol headers and payload fields [10]. Mukherjee et al [31] have also suggested IDS to detect the presence of malware or intruders on a network.

Delivery, propagation and exploit, presented in section 2.3.2.2, are properties connected with the automatic, worm like, distribution of bots. Also these properties can be detected by intrusion detection systems. IDS rules written to detect scanning patterns and executables transmitted on the network will raise alerts. Szor [6] suggested using access control records, described in section 2.5.2.1, to control the network traffic. Communication attempts to unauthorized parties will raise alerts or create log entries of the incidents.

Obfuscation, presented in section 2.3.2.3 refers to techniques used to hide the actual content of what is transmitted across a network. Bots using obfuscation techniques specifically try to evade IDS detection and to pass firewalls. Access contol lists [6] will prevent communication with unauthorized parties, even if the traffic looks benign, and alerts can be raised if the rules are broken.

Botnet server localization was presented in section 2.3.2.4, and refers to methods used by bots to determine the address of the botmasters command and control server. Bots will always have to contact a home server or peer-to-peer node to be able to join a botnet. This activity will always be possible to discover, either automatically via IDS or manually by inspecting a traffic dump of the network activity.

2.5.3.3.Rootkit Detection

Section 2.4.1.1 and 2.4.1.2 presented user-mode and kernel-mode rootkit technologies respectively.

Hoglund and Butler [11] suggest detecting rootkits by observing behaviour. If it is possible to determine if registry keys, files, processes or drivers are hidden from the user, one can conclude that some stealth software is used. Wang et al [27] presented cross-view diff, which is described in section 2.5.1.2. In [11] it is also suggested a rootkit detection method referred to as “guarding the doors”. Since all software must run in memory somewhere, one can look in the memory for malicious rootkit code to determine when it is loaded into memory. Section 2.5.1.4 described monitoring events, proposed by Moffie et al [29]. When a rootkit is about to be installed, it is vulnerable as the stealth functionality is not yet loaded. Monitoring the events performed during installation can be utilized in a “guarding the doors” principle to detect unwanted interaction with kernel modules or user-mode libraries.

Cross-time diff, presented in section 2.5.1.3, was suggested by Wang et al [27] and Kim et al [28] to detect the first versions of user-mode rootkits. If Trojan versions of known applications are used to hide files, ports or processes, an integrity checker would be able to detect this, given a clean baseline is used as comparison.

Yet another technique presented by Hoglund and Butler [11] for discovering rootkits is to look for hooks. Hooking means interfering with a normal data execution flows either by interfering with library calls, system calls or data flows in the driver stack. These rootkit techniques were discussed in section 2.4.1.1 and 2.4.1.2. Libraries and kernel modules offer APIs to make the software environment extensible. Tables in the different libraries contain addresses to all API calls or services offered. These tables are imported by executing applications, so the right API or service can be addressed. By manipulating an address or service-table a rootkit can manipulate data flows to be passed to its own code before returned to the original receiver. There are numerous alternatives for where a hook can be placed. The detection procedure can be like finding a needle in a hay stack. There are however some entry points more used than others, and these are worth monitoring.

3. LABORATORY ENVIRONMENT

In this chapter the laboratory equipment is described, and the software used to build the environment and the tools used in the analysis process are presented. Properties of the chosen analysis utilities will be connected to section 2.5, which explained malware detection. A further presentation of how the laboratory is set up and where the different tools are installed is presented in chapter 3.3.

3.1. Hardware

The hardware used in this test setup is superfluous equipment borrowed from SINTEF ICT and NTNU, and representative for what may exist in stores within different companies.

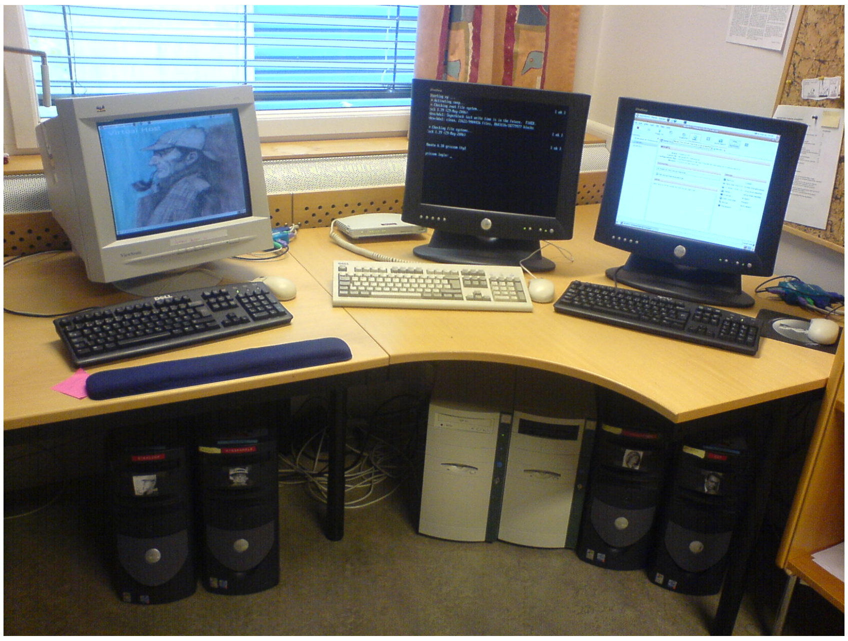


Figure 6: Laboratory environment at SINTEF ICT.

Figure 6 show the laboratory equipment in use. The computers from left are: Sherlock, Marple, Grissom (used as an external storage device), Horatio, Drew and Bond. Table 1 illustrates the main specification of the different computers. In addition to what is presented in the table, each computer is equipped with an 80GB, 7200 rpm hard disk. Grissom is not part of the analysis environment, and therefore not further described.

Computer name	System type	Processor	RAM	NIC
Sherlock	Dell GX260	Pentium 4 2GHz	512MB	3 x 10/100 Mbps
Marple	Dell GX260	Pentium 4 2GHz	512MB	2 x 10/100Mbps
Drew	Dell GX260	Pentium 4 2GHz	512MB	1 x 10/100Mbps
Bond	Dell GX260	Pentium 4 2GHz	512MB	1 x 10/100Mbps
Horatio	Custom Built	AMD Athlon 1,4GHz	1024MB	1 x 10/100Mbps

Table 1: Computer hardware specification

From the table one can see the main limitation in capacity is memory. As further explained later, the laboratory is based on virtual machine technology. Each virtual machine will share hardware resources. A modern operating system should be run with at least 256MB RAM, thus most of the machines above should not handle more than one host operating system and one virtual guest system. At the most extreme, the whole laboratory environment could be reduced to one well equipped computer as long as virtual machine technology is used.

Each machine is physically connected to the laboratory network via a 3COM OfficeConnect 10/100MB 8 Port hub. Three 2-port KVM switches are used to reduce the amount of I/O devices needed.

3.2. Software

As previously mentioned, the software used to build the laboratory environment and the utilities used in the analysis process are all freely available with one exception; since the objects to be studied is designed for the MS Windows platform, one license of MS Windows XP is required.

3.2.1. Basic Installation

Every computer in the network is configured with the same basic setup, which consists of an operating system, virtual machine software, a system hardening tool and an integrity checker. In addition a GUI tool is installed to configure the built-in Linux firewall.

Ubuntu 6.10 Edgy Eft was chosen as the host operating system. Ubuntu is a Linux distribution developed with user-friendliness and security in mind. It is based on Debian GNU/Linux and the project has stated that Ubuntu will always be free of licence cost. Security patches are distributed on a regular basis and free support is offered. Network access can be configured using the built in iptables/netfilter [Web19] packet filtering framework.

VMware Server 1.0.2 [Web20] was chosen as the virtualization system. VMware Server is a free product and offers the flexibility needed in a malware analysis scenario. It is required to agree to a license agreement and register at VMware before using this product. With this tool it is possible to create several virtual machines running in parallel inside the host operating system. Virtual machines and the host operating system share hardware resources, yet the virtualization technology prevent unauthorized interaction between virtual machines, and between virtual machine and the host system. This property is highly desirable when dealing with malware.

Bastille Linux [Web21] is a system hardening tool proactively configuring an operating system with increased security. By “locking down” the operating system, Bastille will decrease its susceptibility to compromise. A series of security related questions are presented to the user when Bastille is run, among others who should have access to compilers and which services should be offered locally and remotely. Each question has a well documented description of consequences related to the different answers. With Bastille’s help, the most common techniques used by malware to infect a Linux system are eliminated from the list of possible attacks. Bastille Linux is free and published under the GNU/GPL license terms.

Samhain [Web22] file integrity/intrusion detection system is used to further control the base of the laboratory environment. Samhain can be configured to keep status of files, processes, open ports and kernel modules, as the most important, in addition to other aspects. Checksums (TIGER192, SHA-1 or MD5) of the interesting objects are kept either locally or centrally and compared at a regular basis, or on demand. This technique corresponds to the *cross-time diff* approach explained in section 2.5.1.3. An integrity check of the kernel will make it possible to discover kernel-mode rootkits in Linux environments. Samhain is published under the GNU/GPL free software license.

Firestarter [Web23] offers an intuitive graphical user interface for configuration of the Linux firewall iptables. Network address translation, packet forwarding and packet filtering are easy to configure using this tool. GNU/GPL licensing is used to distribute this free utility.

3.2.2. Test Environment – Analysis Tools

In this section a presentation of the analysis tools used in the following experiments are described. Several tools used for host-based analysis of the Windows platform are provided free of charge from Microsoft Sysinternals, and a brief introduction is therefore appropriate.

The Sysinternals web site [Web28] was founded in 1996 by Russinovich and Cogswell. They used the site to publish their self developed, advanced system utilities for Windows, including the source code, for free. In July 2006 Microsoft acquired Sysinternals. The system tools are still free to download and use, but regulated by a license agreement. The tools' source code is no longer provided.

3.2.2.1.Hosed-Based Analysis – Auto-Starting Extensibility Points

Section 2.5.1.1 described the theories presented by Wang et al [26], where auto-starting extensibility point was suggested as variables to study when looking for malicious functionality.

AutoRuns for Windows [Web24] is a utility with comprehensive knowledge of auto-starting locations on the MS Windows platform. This utility is provided by Windows Sysinternals. The program can be configured to monitor locations that applications can use to start automatically either after reboot or together with MS Internet Explorer. Registry entries and start-up folders are typical locations that malware edits to survive reboots. Extensibility points offered by Internet Explorer, originally intended for benign toolbars or browser helper objects, are also popular places to manipulate. All of these points are monitored by AutoRuns.

3.2.2.2.Host-Based Analysis – Cross-View Diff

Cross-view diff was presented in section 2.5.1.2. Wang et al [27] presented a theory where the idea was to compare results gained from requests to the operating system at two different levels.

RootkitRevealer [Web25] from Sysinternals is a utility designed after the principles proposed in [27]. RootkitRevealer tries to find discrepancies in lists of files and registry settings collected from high-level API calls and low-level information gathered directly from the raw file system and registry. This utility only tries to find hidden files and registry settings, thus it is not able to detect malware that hides processes using the DKOM techniques explained in section 2.4.1.2.

3.2.2.3.Host-Based Analysis – Cross-Time Diff

Cross-time diff was presented in section 2.5.1.3. Wang et al [27] presented a theory where the idea was to compare the state of the operating system at two different times.

Osiris [Web45] is a file integrity checker, which can be configured to monitor one single host or several hosts in a network. Once the baseline of a clean system is set, Osiris can monitor changes to chosen files and directories. Hash-values (MD-5 or SHA-1) of monitored objects are created to be able to discover modifications of files or folder content. The source code is provided and the application is free to use and modify as long as the terms in the application's license agreement is followed.

3.2.2.4.Host-Based Analysis – Event Monitoring

In section 2.5.1.4, the details of event monitoring was explained. Moffie et al [29] suggested event monitoring techniques to detect malicious behaviour. API calls, file activity, process activity and the like can be watched to discover unwanted functionality.

Process Explorer [Web26] is yet another tool from Windows Sysinternals. It is a nice graphical tool used to display currently running processes. Each process' resource usage is displayed, and a list of which libraries, files or registry keys it has opened can be seen. As soon as a new process is created, it will be shown in Process Explorer.

Process Monitor [Web27], also a Windows Sysinternal tool, is an advanced utility for Windows. It is able to show all file, registry and process/thread activity on a computer in real-time. A filter module makes it possible to only display activity of interest. The activity can be logged to a file for in-depth analysis at a later time.

These tools have overlapping functionality. Together they will make a pretty complete picture of what events took place during execution of given applications.

3.2.2.5.Host-Based Analysis – Rootkit Detection

Rootkit detection techniques were studied in section 2.5.3.3. Hoglund and Butler [11] suggested looking for API hooks i.e. look for places where rootkits intercept regular dataflows. Together with Wang et al [27], they also suggest using *cross-view diff* approaches to discover rootkits.

RootkitRevealer from Sysinternals follows the cross-view diff approach. A further description is found in section 3.2.2.2.

Gmer [Web46] is a rootkit detection tool utilizing both cross-view diff principles to detect hidden files, processes, registry settings, services and drivers. The tool also scan the host for API hooking applications. Gmer is freely distributed.

3.2.2.6.Network-Based Analysis – Access Control, IDS, Honeypots

In section 2.5.2, network-based malware detection techniques were introduced. Szor [6] described access control via router access lists, Mukherjee et al [31], among others, suggested IDS solutions and Spitzner [8] presented honeypot technology to study and fight the attackers.

The Honeynet Project [Web30], where Spitzner is president, is a non-profit volunteer research organization. Their goal is to improve security on the Internet at no cost to the public. Several honeypot tools are available through the project, among them the high-interaction honeypot tool Honeywall Roo. This tool combines all the functionality presented in [6, 8, 31]. Every utility presented next is published under the GNU/GPL license.

Honeywall Roo is a preconfigured environment where several tools are included to be able to monitor attacker's behaviour on a compromised system. When installing Honeywall, the chosen computer is turned into a layer-two bridge. Network traffic is passed through the bridge and since it is operational on the data link layer, attackers will have difficulty detecting its presence. A Honeywall is typically placed between a honeypot and the Internet to be able to intercept all traffic sent to and from a compromised host/honeypot. All network activity is automatically logged and stored in a database. Honeywall Roo is built on the Fedora Core Linux distribution. Data capture and analysis tools included in the Honeywall environment are Snort, Oinkmaster and Walleye, among others. Walleye is a web interface used to present and analyze captured data. The other tools are presented next.

Snort [Web31] is an open source intrusion detection system. IDSs were presented in section 2.5.2.2. Snort detection is built on a rule driven language. Detection of unwanted network activity can be based either on signature-based approaches, protocol or anomaly detection. The rule set is flexible and it is easy to include self written rules. A traditional intrusion detection system will inspect network traffic and raise alerts if a given rule is matched. Snort can operate in a mode called Snort Inline. Running in this mode Snort can act as an intrusion prevention system (IPS). The difference between IDS versus IPS is that where an IDS will raise alerts, but still allow the traffic to pass, an IPS can actively block the traffic. Snort is configured in Inline mode in the Honeywall configuration.

Oinkmaster [Web32] is a perl script designed to automate the process of updating and managing Snort rules.

In addition to the tools described above, which automates the process of detecting unwanted network traffic, one additional tool is used. To be absolutely sure no events pass by unnoticed; a packet sniffer logging all network activity, in addition to the Honeywall, is used. This will make it possible to manually inspect every packet sent on the network.

Wireshark [Web33] is a free network protocol analyzer published under the GNU/GPL license. It is able to collect every packet of data sent on network. Traffic flows can be analysed in a nice, intuitive graphical user interface. It can also read data from a variety of platforms like token ring and Ethernet, among others. Ethereal has an extensive list of protocols, which it can dissect. Users can easily configure own display filters for the captured traffic.

3.2.2.7.Network-Based Analysis – Remote Vulnerability Checking

Remote vulnerability checking was presented in section 2.5.2.4. Skaggs et al [34] proposed this method to detect unwanted functionality on a host. Two different utilities are used to scan for remote vulnerabilities.

Nmap [Web34] is a free open source utility for network exploration and security auditing, and follows the terms of GNU/GPL licensing. It is designed to scan large networks in a short amount of time. It is easily configured to scan given hosts or entire network segments. Nmap can return reports on which hosts are available, which services they are offering and which operating system the remote hosts are running on.

Nessus [Web35] is a remote vulnerability scanner. Updated signature files of known vulnerabilities are provided by the developers of Nessus and can be installed as plug-ins to the program. Nessus does scanning of remote machines or entire networks. Initially the application starts out by doing a port scan of the target. Next, probes of known attack patterns are sent to see if there is any response. Based on the collected data, reports of security related importance is generated. As is the case with every other signature based system, like for instance Anti-Virus software, detection of security holes with Nessus is dependent on the presence of a matching signature rule set. Nessus is regulated by a license agreement, and one has to register at the web site before the tool can be used. Access to the latest Nessus security plug-ins can be bought. However, all plug-ins are distributed for free with a 7 days delay. The tool itself is free to use.

3.2.2.8.Malware Detection – Traditional Anti-Virus

The focus of this work is to analyze software behavior. To be able to determine which exact malware specimens were studied, a traditional anti-virus tool was used. An anti-virus scan was performed on every application studied after the behavioral tests to see if the conclusions regarding the test objects were right.

ClamWin [Web36] is a free open source anti-virus tool for the MS Windows platform published under the GNU/GPL license. Virus scans are started on demand. It is a signature-based detection engine, and updated signature files can be automatically downloaded.

3.3. Design and Implementation of Test Environment

In this chapter the laboratory environment is described, and arguments of the design principles are given. Applications in use are presented in *italic*, and their descriptions are found in section 3.2. Figure 7 illustrates the current setup.

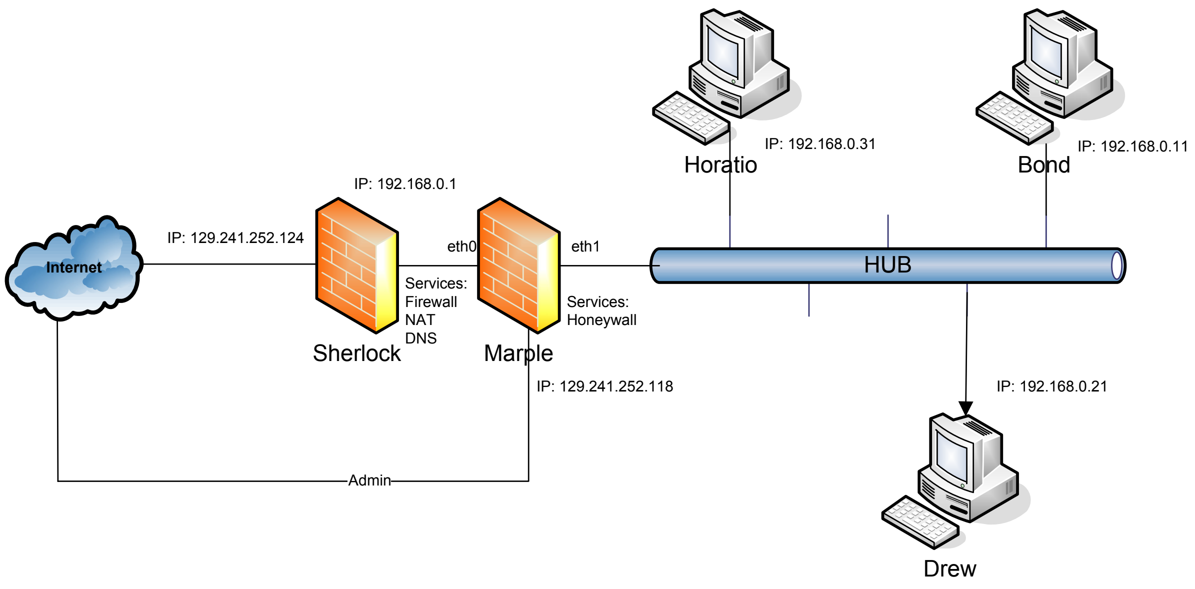


Figure 7: Realization of the laboratory environment.

This picture illustrates how the laboratory environment is realized. Sherlock act as the firewall, Marple is configured with Honeywall software, Horatio act as a remote vulnerability scanner, Drew is a network sniffer and Bond is the victim to observe.

The laboratory is based on virtual machines because of the flexibility offered by virtualization technology. When dealing with malware, the risk of infecting the entire laboratory environment is present. With *VMware Server*, used in this setting, each virtual host can be reverted to a previously known clean state in seconds. In comparison, formatting a hard drive and installing an operating system with necessary applications from scratch can take more than an hour.

Natvig [4], among others, has explained the risk of malware escaping virtual environments. The initial laboratory environment designed in fall 2006 [12] was, like the current setup, based on virtual machine technology. The operating systems in which the virtual machines were run was, however, MS Windows 2000. Both the work presented in this report and the previous work, had a focus on analyzing malicious code written for MS Windows systems. The risk of infecting the entire laboratory was therefore present in the previous setup. The probability of malware being able to infect both Windows and Linux system is relatively low, though present. To reduce the risk of malware escaping the virtual environment and infecting the host operating system, *Ubuntu* Linux is installed as a base operating system in the laboratory.

To further reduce the risk of unwanted infections, *Bastille* is used to harden the Linux environment. Administrative tools are restricted to root, and services not necessary are disabled. The configuration of *Bastille* can be found in appendix B. *Samhain*, the file integrity checker, is used to control important operating system files and the kernel-module for infections. *Samhain* is also configured to control the integrity of the virtual machine files. This is to ensure an uninfected version of the virtual guest OS is run for each test.

In addition to making the laboratory more secure from being unintentionally infected, one major improvement is made to the setup presented in [12] and Figure 1. This improvement involves the use of *Honeywall Roo*. The H*oneywall* is installed on Marple and was motivated by a malware trend presented in [Web1], which explained the use of staged downloads of malware.

In an isolated, sandbox like environment, only the installation file distributed in the first stage of the malware replication could have been tested for unwanted functionality. Malicious content can very well be hidden in the code downloaded during the installation process. It is a realistic scenario that software repositories can be compromised and benign software replaced with Trojan versions. A more secure way of connecting to Internet than giving full access, is thus needed to provide more flexibility. In a worst case scenario e.g. the Google Desktop installer file could be a Trojan horse modified to download a bot binary. This bot could start infecting other machines instantly or force the laboratory computer to participate in a DDoS attack.

The *Honeywall* is installed as a layer-two bridge inserted between the hosts in the laboratory setup and the firewall. This ensures every single data packet directed towards Internet is inspected before they reach the net. As explained in the previous chapter, *Honeywall Roo* includes the IDS *Snort* in inline mode. This means it can block the Internet connection if malicious network traffic is found. For instance, the replication attempt of a worm can be detected and blocked. The Honeywall can also be set up to allow a maximum number of connections per time unit. Internet access is blocked if the number of connections are exceeded. By modifying these parameters, one can prevent worms from scanning for new vulnerable hosts, or prevent laboratory computers from being participants in DoS attacks. From Figure 7 one can see where the *Honeywall* is placed. The introduction of the high-interaction honeypot technology increases the flexibility of the behavioural analysis environment considerably.

Iptables is used to configure Sherlock to become a firewall with NAT functionality. The firewall is controlled graphically via *firestarter*. It can be configured to allow separate hosts Internet access or Internet access based on communication protocols. By default, in this setup, the firewall is configured to block all inbound and outbound access. NAT is used to give the laboratory machines private, un-routable IP addresses. This is to make certain they do not communicate with other machines outside the test environment via other channels than through Sherlock, the NAT gateway. The honeywall has an administrative network interface which, even though unlikely, could be used by attackers to escape the laboratory environment.

Sherlock is also equipped with its own DNS service. Some malware attack the DNS system. By controlling and giving access only to an internal DNS server one ensures production systems are not attacked. This setup also allows laboratory computer access to DNS services without any direct communication with servers outside the test network.

Iptables controlled through *firestarter*, ensures network connections to each Linux host in the network is rejected, thus infections through network replication are avoided.

It is crucial to keep the risk of each test as low as possible. An infected laboratory machine being able to compromise computers outside the laboratory environment can lead to serious legal consequences.

Computer Name	Host Operating System	Virtual Guest Operating System
Sherlock	*Ubuntu Edgy Eft* - *VMware Server* - *Bastille* - *Samhain* - *Firestarter* - *Bind9, DNS server*	*Ubuntu Edgy Eft* - *Bastille* - *Samhain* - *Firestarter*
Marple	*Ubuntu Edgy Eft* - *VMware Server* - *Bastille* - *Samhain* - *Firestarter*	*Honeywall Roo* - *Snort* - *Oinkmaster* - *Sebek server*
Horatio	*Ubuntu Edgy Eft* - *VMware Server* - *Bastille* - *Samhain* - *Firestarter*	*Ubuntu Edgy Eft* - *Bastille* - *Samhain* - *Firestarter* - *Nmap* - *Nessus*
Drew	*Ubuntu Edgy Eft* - *VMware Server* - *Bastille* - *Samhain* - *Firestarter*	*Ubuntu Edgy Eft* - *Bastille* - *Samhain* - *Firestarter* - *Wireshark*
Bond	*Ubuntu Edgy Eft* - *VMware Server* - *Bastille* - *Samhain* - *Firestarter*	Microsoft Windows XP SP2 - *ProcessMonitor* - *ProcessExplorer* - *RootkitRevealer* - *AutoRuns* - *Osiris* - *Gmer*

Table 2: Overview of operating systems and tools installed on the different computers.

Table 2 illustrates which operating systems and tools are installed on the different machines in the laboratory environment. During the project assignment of fall 2006 [12], three roles were identified and should be covered in a network-based detection scenario; a victim, an eavesdropper and an attacker. The victim is the host to be analyzed and where the object to be scrutinized should be installed. The eavesdropper should silently listen in to all traffic to be able to determine if any malicious data is sent. The attacker should actively probe the victim for vulnerabilities by simulating attacker behaviour. In the laboratory setup presented in Figure 7, Bond is the victim, Drew and Marple are the eavesdroppers and Horatio is the attacker. For a further reasoning behind the roles, it is referred to [12].

Drew and Marple have overlapping functionality. Both record all data activity on the network. Drew could be removed without loss of functionality. However, this computer is kept in the laboratory setup for security reasons. As figure 7 indicates, the data flow to Drew goes one way. Appendix E sketches a receive-only Ethernet cable used connect this computer to the network. By physically cutting the transmitting wires, Drew is physically unable to communicate with the other laboratory computers and vice versa. This will eliminate the risk of this machine of being infected by automated infection procedures. Still, the receive-only cable allows sniffing data from the network. In a worst case scenario where every other machine in the laboratory environment became infected and log files deleted, Drew would contain an image of the contamination phase.

When setting up a host-based analysis environment, the malware detection theory presented in section 2.5.1 was deciding for which properties should be included in the analysis tools. Each theory presented represents different approaches to malware detection. None of the existing tools found, include properties from all malware detection approaches. This is why several tools are used, each containing interesting elements for a behavioural analysis.

4. EXPERIMENTS AND RESULTS

The experiments performed during this work were carried out with bot code caught in the wild. The rootkit analyzed was, however, downloaded from [Web39] and is an example of how a pure rootkit may be implemented.

In the following a brief introduction to how the malware was collected is given. Next, one detailed example of behavioural software analysis is presented. This will introduce how the experimentation and analysis phase was performed. The most important findings of the other tests are summarized at the end of this chapter.

4.1. Preparation – Collecting Malware

Three different sources were used to acquire bot code; the file-sharing network Kazaa, a medium-interaction honeypot called Nepenthes and a contribution from Einar Oftedal at NorCERT. The fourth malware source was, as mentioned, [Web39] the web site of Hoglund and Butler.

The introduction to chapter 1 presented the work of Shin et al [1], where malware prevalence in the Kazaa file-sharing network was explored. They discovered that by using the search phrase “ICQ”, the chances of hitting downloadable files infected with malware were more than 70%. This result was used as motivation for the use of Kazaa as a source for downloading active malware.

Kazaa [Web44] was installed on Bond in the laboratory network presented in chapter 3.3. Once Kazaa was installed and Bond connected to the Kazaa network, “ICQ” and “WinZip” was used as search phrases. Every ICQ and Winzip application found was downloaded. It turned out that seven out seven downloaded applications were Trojan horses. The downloaded applications were named:

- ICQ hacks.exe
- ICQ Lite (new).exe
- ICQ Lite beta (b2253).exe
- ICQ Pro 2003a.exe

- ICQ Pro 2003a beta (b4600).exe
- ICQ Pro 2003b (new beta).exe
- Winzip_Crack.exe

This simple experiment was only used to get the hands on malware currently active on the Internet. It is not meant to contribute to the work of Shin et al or to be concluded in any way. It is, however, interesting to see that the threats of downloading infected files from file-sharing networks are real.

More malware was collected through use of the medium-interaction honeypot Nepenthes [Web40]. Nepenthes is a honeypot sensor that simulates several different vulnerable services, which are exploited by auto-replicating malware. The sensor is connected to Internet, and once an exploit attempt is made on one of the emulated services, a download handler tries to download the malware that initiated the connection. A hash value of each downloaded file is calculated to ensure each malicious specimen is only stored once. Several connection attempts were made towards the sensor, but only four successful unique downloads were made through a two months period. The names of the downloaded files were:

- Internat.exe, caught twice
- Winlist.exe, caught twice

Internat.exe and Winlist.exe were both downloaded twice, but with different hash values. Different hash values may indicate that the malware bodies of Internat.exe and Winlist.exe have mutated. Except for these comments, other aspects of the Nepenthes experiments are not considered.

The last source of malicious bot code was Einar Oftedal at NorCERT who contributed to this work with two bot-binaries; one IRC based and one peer-to-peer based, called bot.exe and via.exe respectively.

In addition to the bot code presented, the behaviour of one rootkit, HackerDefender [Web6], is analyzed. Some background knowledge of this rootkit is useful when looking at the following test results.

HackerDefender is a user-mode rootkit capable of hiding processes, files, folders, registry settings and communication ports. The rootkit hooks into system libraries to be able to intercept and manipulate data-flows between applications and the operating system.

Backdoors are created by hooking into every previously opened communication port to be able to hide from both local and remote port listing tools. Specially crafted HackerDefender packets sent to each port are handled by the rootkit. Every other packed is forwarded to the service that originally opened the port.

4.2. Experiments

In this section, one experiment is presented in detail. This is to demonstrate how the laboratory environment, that is designed based on the dynamic behavioural malware detection theory presented in section 2.5, can be used to discover unwanted functionality in software. Each test is performed following the same routines, which are described in appendix C, to make sure the results are comparable.

ICQ Lite (new).exe is a Trojan horse downloaded from the Kazaa file-sharing network and will be used in the following case to demonstrate the usefulness of a behavioural analysis. The motivation behind this work was given in chapter 1. It was stated that to run applications downloaded from unknown sources may be risky. Traditional anti-virus detection schemes have limitations when it comes to the ability to detect previously unknown malware. Thus, it may be difficult to determine whether applications are manipulated by third parties to contain unwanted functionality or not, by the use of traditional detection tools. ICQ Lite (new).exe is an example emphasizing the worries addressed in the introduction to this report.

ICQ is a freely available instant messaging application with an official home page [Web42] where the application can be downloaded. File-sharing networks, however, are popular and many prefer these file-sharing points to download their software. By its name, ICQ Lite (new).exe looks like a benign application that very well could have been an official software distribution. A person eager to be part of the ICQ instant messaging community, may download ICQ Lite (new).exe ignorant of the fact that it really is a Trojan horse concealing a malicious bot with replication characteristics of peer-to-peer worms, which will be demonstrated next.

After a double-click on the newly acquired application, one would expect an installation procedure to start. However, nothing seemed to happen after activating the executable. An unskilled/ignorant user would probably conclude that the file was broken and try new downloads. The next sections will demonstrate how the behavioural analysis laboratory can be used to determine what really happens after files are opened.

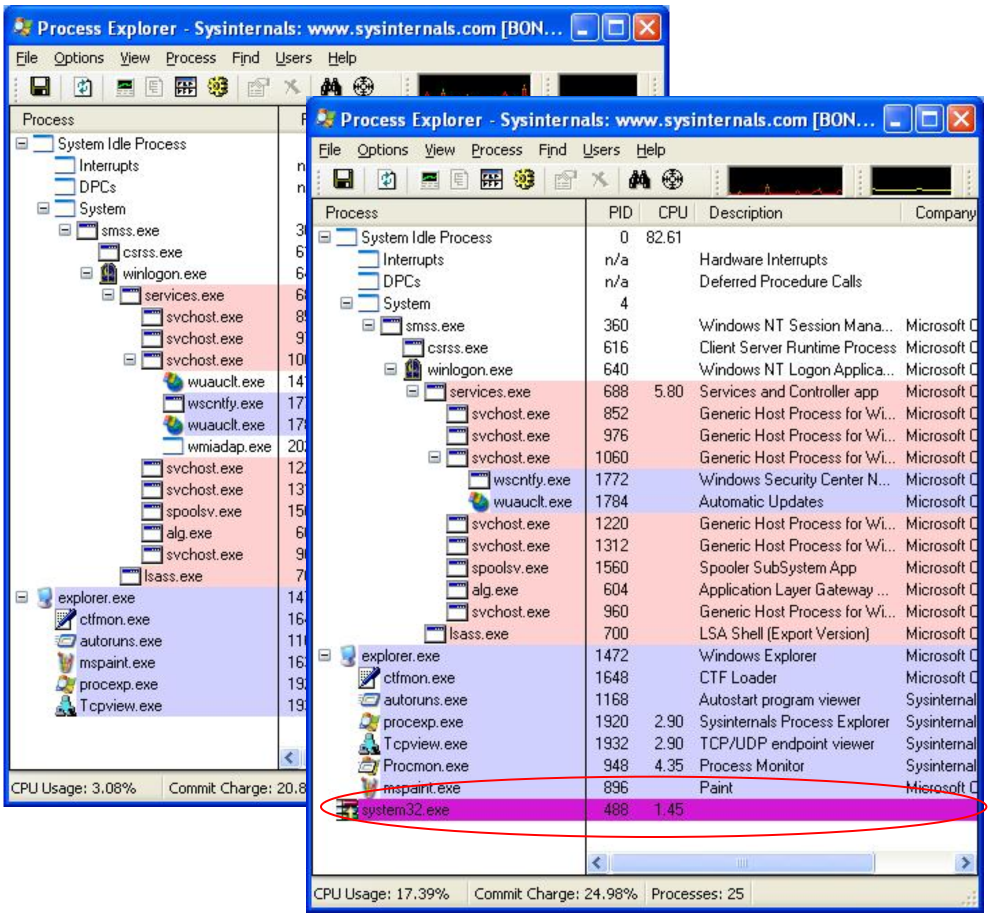
Each of the experiments performed in the laboratory were conducted twice. The first run was performed without any connection to Internet. The firewall, Sherlock, was configured to block all traffic to and from the laboratory network. Data was logged for five minutes after activation of the objects to analyze. In the second run, the same procedure was followed. However, after five minutes had passed, the firewall was configured to allow DNS traffic. Data was recorded for three more minutes. The laboratory setup contains its own DNS server. The victim computer was never allowed direct connection with any computers outside the laboratory environment.

4.2.1. Host-Based Behavioural Analysis

Six different tools were used locally on Bond to capture behaviour related data. *AutoRuns, Osiris, ProcessExplorer, ProcessMonitor, RootkitRevealer and Gmer*, as explained earlier.

4.2.1.1.Event Monitoring – ProcessExplorer

ProcessExplorer gives a graphical representation of each running process in real time. Observing process activity fall under the event monitoring principles of Moffie et al [29]. By having this utility up and running, one can see which processes are opened and closed by highlighted colours. Currently running processes are also displayed. Observing the output of this application gave the first indication of the installation of suspicious functionality. As mentioned earlier, nothing seemed to happen when the ICQ Lite application was run. *ProcessExplorer*, however, reported the creation of a running process.



Process Explorer - Sysinternals: www.sysinternals.com [BON...

File Options View Process Find Users Help

```
Process
System Idle Process
  Interrupts          n
  DPCs                n
  System
    smss.exe          3
      csrss.exe       6
      winlogon.exe    6
        services.exe  6
          svchost.exe 8
          svchost.exe 9
          svchost.exe 10
            wuauclt.exe   14
            wscntfy.exe   17
            wuauclt.exe   17
            wmiadap.exe   20
          svchost.exe 12
          svchost.exe 13
          spoolsv.exe 15
          alg.exe     6
          svchost.exe 9
        lsass.exe     7
explorer.exe          14
  ctfmon.exe          16
  autoruns.exe        11
  mspaint.exe         16
  procexp.exe         19
  Tcpview.exe         19

CPU Usage: 3.08%   Commit Charge: 20.8
```

Process Explorer - Sysinternals: www.sysinternals.com [BON...

File Options View Process Find Users Help

Process	PID	CPU	Description	Company
System Idle Process	0	82.61		
Interrupts	n/a		Hardware Interrupts	
DPCs	n/a		Deferred Procedure Calls	
System	4			
smss.exe	360		Windows NT Session Mana...	Microsoft C
csrss.exe	616		Client Server Runtime Process	Microsoft C
winlogon.exe	640		Windows NT Logon Applica...	Microsoft C
services.exe	688	5.80	Services and Controller app	Microsoft C
svchost.exe	852		Generic Host Process for Wi...	Microsoft C
svchost.exe	976		Generic Host Process for Wi...	Microsoft C
svchost.exe	1060		Generic Host Process for Wi...	Microsoft C
wscntfy.exe	1772		Windows Security Center N...	Microsoft C
wuauclt.exe	1784		Automatic Updates	Microsoft C
svchost.exe	1220		Generic Host Process for Wi...	Microsoft C
svchost.exe	1312		Generic Host Process for Wi...	Microsoft C
spoolsv.exe	1560		Spooler SubSystem App	Microsoft C
alg.exe	604		Application Layer Gateway ...	Microsoft C
svchost.exe	960		Generic Host Process for Wi...	Microsoft C
lsass.exe	700		LSA Shell (Export Version)	Microsoft C
explorer.exe	1472		Windows Explorer	Microsoft C
ctfmon.exe	1648		CTF Loader	Microsoft C
autoruns.exe	1168		Autostart program viewer	Sysinternal
procexp.exe	1920	2.90	Sysinternals Process Explorer	Sysinternal
Tcpview.exe	1932	2.90	TCP/UDP endpoint viewer	Sysinternal
Procmon.exe	948	4.35	Process Monitor	Sysinternal
mspaint.exe	896		Paint	Microsoft C
system32.exe	488	1.45		

CPU Usage: 17.39%   Commit Charge: 24.98%   Processes: 25

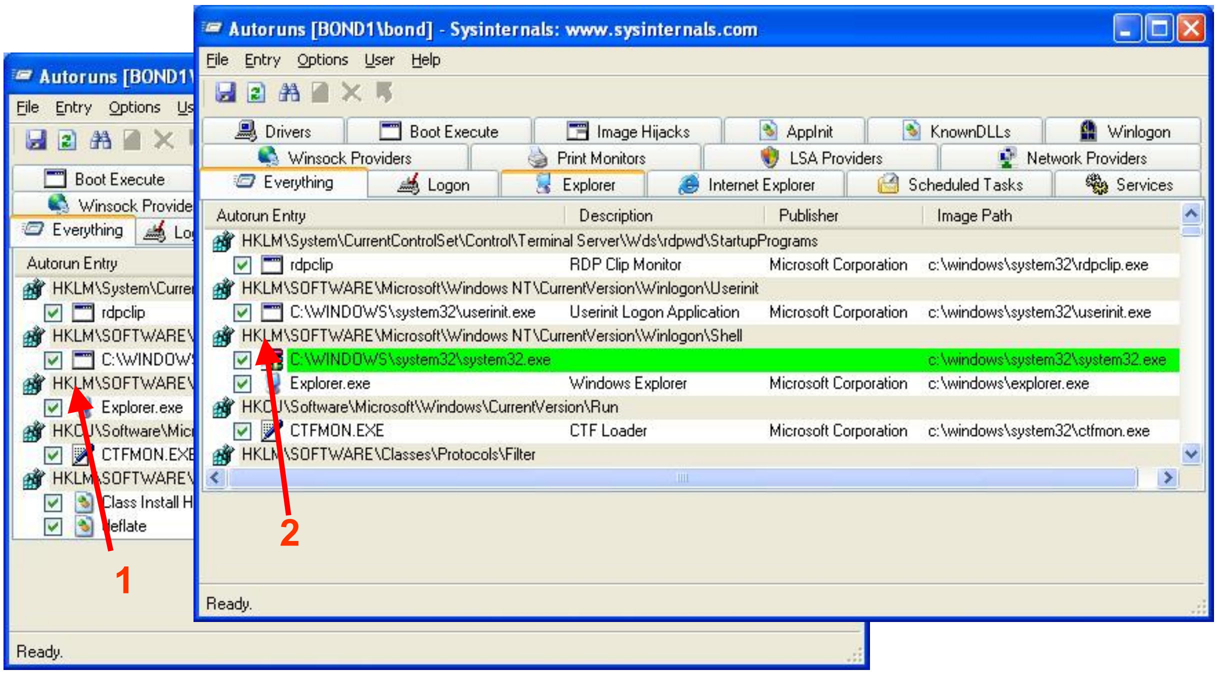
Figure 8: ProcessExplorer displaying running processes.

Figure 8 illustrates the system before and after ICQ Lite was run. The picture in front represents the system after double-clicking the file, while the same system is presented in the background before activation of the application. Even though the user never was presented with any installation information after double-clicking the application, *ProcessExplorer* reported a new active process. System32.exe, highlighted in purple, looked very suspicious. *ProcessExplorer* can be set to highlight different attributes according to predefined rules. Packed processes were chosen to be highlighted by the colour purple. Section 2.3.1.2 explained obfuscation as a principle used by bots, and malware in general, to try to evade signature-based detection, among other tings.

As little could be concluded so far, more tests had to be performed. The process was not proved to be created as a direct consequence of double-clicking the ICQ Lite file.

4.2.1.2.Auto-Start Extensibility Points – AutoRuns

AutoRuns was used to determine if the newly discovered process would try to start up automatically after a reboot. By monitoring the auto-start extensibility points according to the principles suggested by Wang et al [26], *AutoRuns* could give valuable information of which techniques the process would use to survive reboots.



```
Autorun Entry                                                         Description                 Publisher              Image Path
HKLM\System\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\StartupPrograms
    rdpclip                                                           RDP Clip Monitor            Microsoft Corporation  c:\windows\system32\rdpclip.exe
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
    C:\WINDOWS\system32\userinit.exe                                  Userinit Logon Application  Microsoft Corporation  c:\windows\system32\userinit.exe
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
    C:\WINDOWS\system32\system32.exe                                                                                 c:\windows\system32\system32.exe
    Explorer.exe                                                      Windows Explorer            Microsoft Corporation  c:\windows\explorer.exe
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
    CTFMON.EXE                                                        CTF Loader                  Microsoft Corporation  c:\windows\system32\ctfmon.exe
HKLM\SOFTWARE\Classes\Protocols\Filter
```

Figure 9: Auto-start extensibility point discovered with AutoRuns. 1 and 2 points towards the same registry path on a clean and compromised computer, respectively.

Figure 9 illustrates the fact that after the ICQ Lite application was run, a new registry key had been inserted into the path:

- HKLM\SOFTWARE\Microsoft\Windows NT\CurentVersion\Winlogon\Shell

This registry path determines which processes should automatically start when users log into the Windows operating system. On the known clean system only Explorer.exe, the Windows graphical user interface, should be started at login. After activation of ICQ Lite, the System32.exe process was also configured to start automatically.

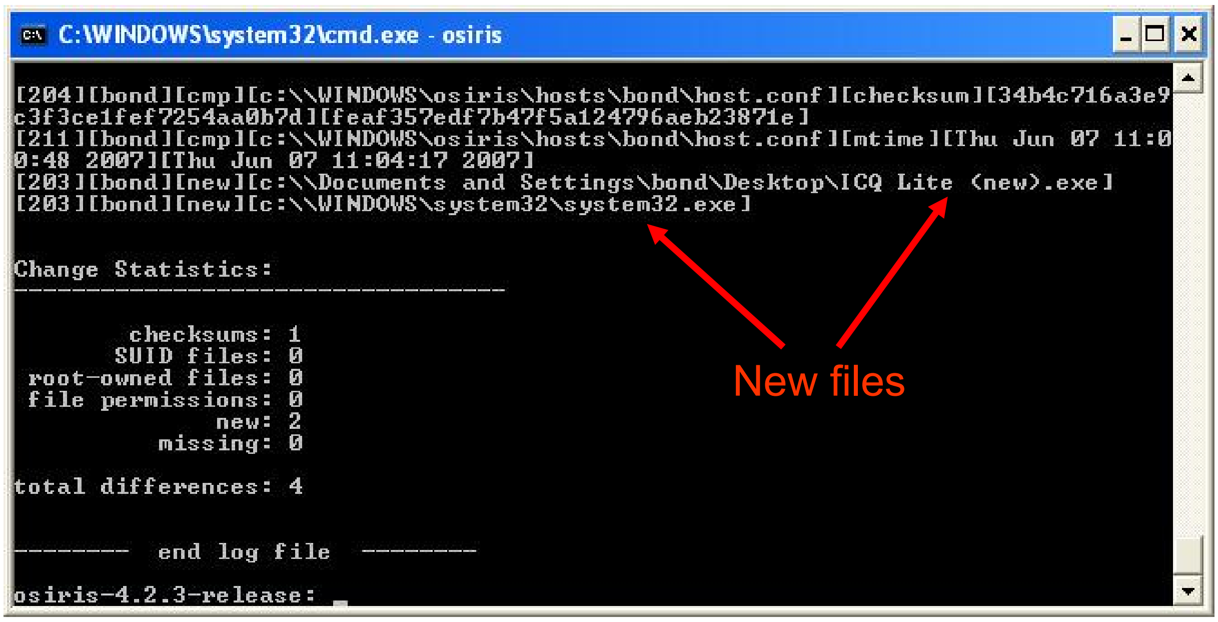
Even though it seemed probable the System32.exe process was created as a consequence of double-clicking ICQ Lite (new).exe, the evidence was still missing.

4.2.1.3.Cross-time diff – Osiris

The registry key added to survive reboots pointed to the file path:

- C:\WINDOWS\System32\System32.exe

To determine whether this file existed before ICQ Lite was run, or if it was new to the system, a file integrity check, using *Osiris*, was performed. This check could even reveal if other new files were created or if existing files had been changed. This test follows the cross-time diff principles suggested by Wang et al [27] and Kim and Spafford [28].



```
[204][bond][cmp][c:\\WINDOWS\osiris\hosts\bond\host.conf][checksum][34b4c716a3e9
c3f3ce1fef7254aa0b7d][feaf357edf7b47f5a124796aeb23871e]
[211][bond][cmp][c:\\WINDOWS\osiris\hosts\bond\host.conf][mtime][Thu Jun 07 11:0
0:48 2007][Thu Jun 07 11:04:17 2007]
[203][bond][new][c:\\Documents and Settings\bond\Desktop\ICQ Lite (new).exe]
[203][bond][new][c:\\WINDOWS\system32\system32.exe]


Change Statistics:
-----------------------------------

        checksums: 1
       SUID files: 0
 root-owned files: 0
 file permissions: 0
              new: 2
          missing: 0

total differences: 4


--------  end log file  --------

osiris-4.2.3-release: _
```

Figure 10: Output of an Osiris file-integrity check.

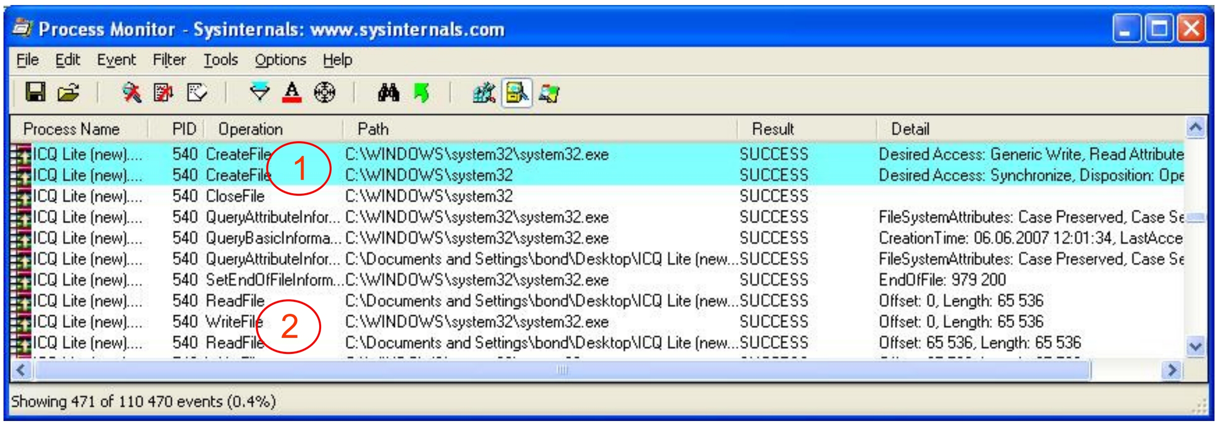
As Figure 10 illustrates, two new files existed on the system after the baseline of a clean system was created. ICQ Lite (new).exe was intentionally copied onto the system to perform the test. System32.exe, however, was new. No[7] other file changing operations than copying the ICQ application had been performed on the computer since the integrity baseline was created. The probability of coherence between the introduction of ICQ Lite (new).exe and the creation of system32.exe was therefore increasing.

[7] The configuration file was changed intentionally before the integrity check, hence the changes to host.conf displayed in Figure 10.

4.2.1.4.Event Monitoring – ProcessMonitor

So far, the event-monitoring principle suggested by Moffie et al[29] was covered by *ProcessExporer*, where the running system32.exe process was first discovered. *ProcessMonitor* is a more comprehensive application and, as described earlier, capable of recording events created by file-, process- and registry activity. This application was set to monitor every event on the system from before the activation of ICQ Lite and for five minutes after.

By studying the *ProcessMonitor* log-file, displayed in Figure 11*,* one could see that the process ICQ Lite (new).exe created the file system32.exe (1). Further, the content of ICQ Lite (new).exe was copied to the system32.exe file (2).

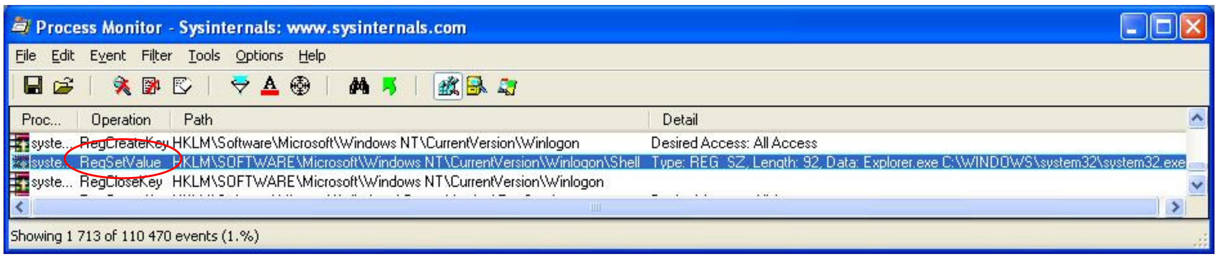

Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
1
2

```
Process Name       PID  Operation           Path                                                 Result    Detail
ICQ Lite (new)....  540  CreateFile          C:\WINDOWS\system32\system32.exe                    SUCCESS   Desired Access: Generic Write, Read Attribute
ICQ Lite (new)....  540  CreateFile          C:\WINDOWS\system32                                 SUCCESS   Desired Access: Synchronize, Disposition: Ope
ICQ Lite (new)....  540  CloseFile           C:\WINDOWS\system32                                 SUCCESS
ICQ Lite (new)....  540  QueryAttributeInfor... C:\WINDOWS\system32\system32.exe                 SUCCESS   FileSystemAttributes: Case Preserved, Case Se
ICQ Lite (new)....  540  QueryBasicInforma... C:\WINDOWS\system32\system32.exe                  SUCCESS   CreationTime: 06.06.2007 12:01:34, LastAcce
ICQ Lite (new)....  540  QueryAttributeInfor... C:\Documents and Settings\bond\Desktop\ICQ Lite (new...SUCCESS   FileSystemAttributes: Case Preserved, Case Se
ICQ Lite (new)....  540  SetEndOfFileInform...C:\WINDOWS\system32\system32.exe                  SUCCESS   EndOfFile: 979 200
ICQ Lite (new)....  540  ReadFile            C:\Documents and Settings\bond\Desktop\ICQ Lite (new...SUCCESS   Offset: 0, Length: 65 536
ICQ Lite (new)....  540  WriteFile           C:\WINDOWS\system32\system32.exe                    SUCCESS   Offset: 0, Length: 65 536
ICQ Lite (new)....  540  ReadFile            C:\Documents and Settings\bond\Desktop\ICQ Lite (new...SUCCESS   Offset: 65 536, Length: 65 536
Showing 471 of 110 470 events (0.4%)
```

Figure 11: Log extract from ProcessMonitor data. 1 shows creation of the system32.exe file and 2 shows file content copied from ICQ Lite (new).exe to the new file.

This was the final proof of connection between the activation of the seemingly corrupted ICQ application and the system32.exe process. The log-file did not show any evidence of more files created, which correlates with the results of the file-integrity check.

Further, the insertion of the registry key leading to the process being able to survive reboots could be found by studying the registry related part of the log-file, illustrated by Figure 12.

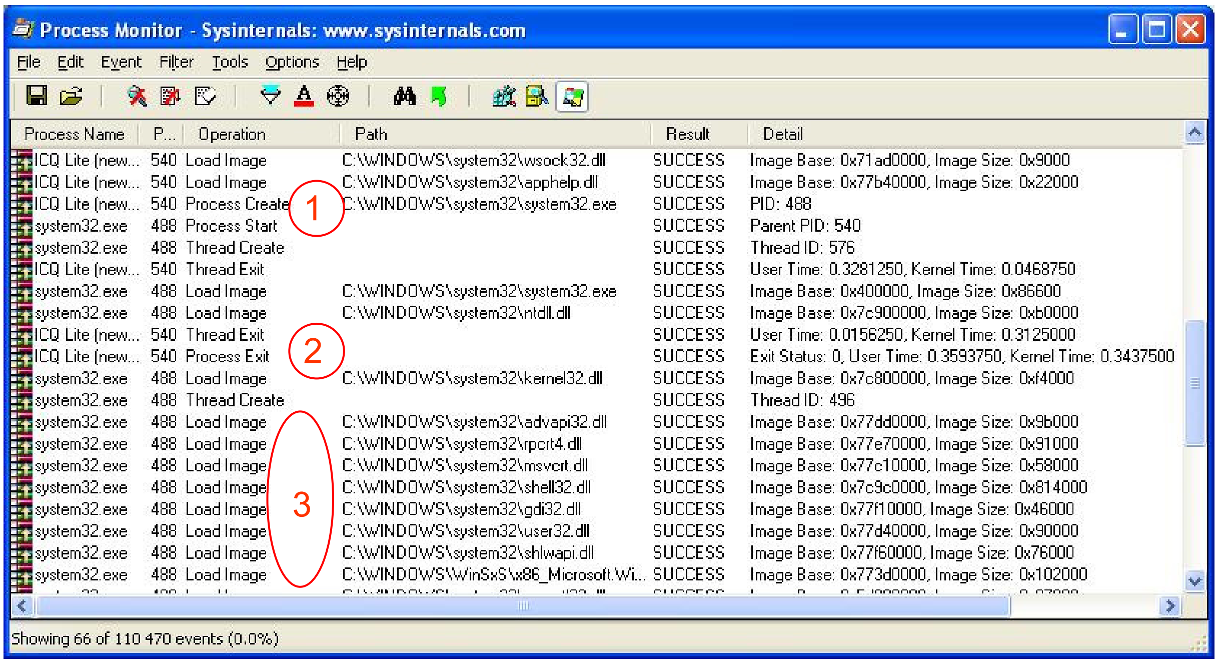

Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Proc...	Operation	Path	Detail
syste...	RegCreateKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon	Desired Access: All Access
syste...	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell	Type: REG_SZ, Length: 92, Data: Explorer.exe C:\WINDOWS\system32\system32.exe
syste...	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	

Showing 1 713 of 110 470 events (1.%)

Figure 12: ProcessMonitor log extract showing insertion (RegSetValue) of the registry key leading to auto-starting properties.

By observing process related events, one could extract more interesting information about the hidden functionality of the ICQ Lite application.


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
1
2
3

Process Name	P...	Operation	Path	Result	Detail
ICQ Lite (new...	540	Load Image	C:\WINDOWS\system32\wsock32.dll	SUCCESS	Image Base: 0x71ad0000, Image Size: 0x9000
ICQ Lite (new...	540	Load Image	C:\WINDOWS\system32\apphelp.dll	SUCCESS	Image Base: 0x77b40000, Image Size: 0x22000
ICQ Lite (new...	540	Process Create	C:\WINDOWS\system32\system32.exe	SUCCESS	PID: 488
system32.exe	488	Process Start		SUCCESS	Parent PID: 540
system32.exe	488	Thread Create		SUCCESS	Thread ID: 576
ICQ Lite (new...	540	Thread Exit		SUCCESS	User Time: 0.3281250, Kernel Time: 0.0468750
system32.exe	488	Load Image	C:\WINDOWS\system32\system32.exe	SUCCESS	Image Base: 0x400000, Image Size: 0x86600
system32.exe	488	Load Image	C:\WINDOWS\system32\ntdll.dll	SUCCESS	Image Base: 0x7c900000, Image Size: 0xb0000
ICQ Lite (new...	540	Thread Exit		SUCCESS	User Time: 0.0156250, Kernel Time: 0.3125000
ICQ Lite (new...	540	Process Exit		SUCCESS	Exit Status: 0, User Time: 0.3593750, Kernel Time: 0.3437500
system32.exe	488	Load Image	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Image Base: 0x7c800000, Image Size: 0xf4000
system32.exe	488	Thread Create		SUCCESS	Thread ID: 496
system32.exe	488	Load Image	C:\WINDOWS\system32\advapi32.dll	SUCCESS	Image Base: 0x77dd0000, Image Size: 0x9b000
system32.exe	488	Load Image	C:\WINDOWS\system32\rpcrt4.dll	SUCCESS	Image Base: 0x77e70000, Image Size: 0x91000
system32.exe	488	Load Image	C:\WINDOWS\system32\msvcrt.dll	SUCCESS	Image Base: 0x77c10000, Image Size: 0x58000
system32.exe	488	Load Image	C:\WINDOWS\system32\shell32.dll	SUCCESS	Image Base: 0x7c9c0000, Image Size: 0x814000
system32.exe	488	Load Image	C:\WINDOWS\system32\gdi32.dll	SUCCESS	Image Base: 0x77f10000, Image Size: 0x46000
system32.exe	488	Load Image	C:\WINDOWS\system32\user32.dll	SUCCESS	Image Base: 0x77d40000, Image Size: 0x90000
system32.exe	488	Load Image	C:\WINDOWS\system32\shlwapi.dll	SUCCESS	Image Base: 0x77f60000, Image Size: 0x76000
system32.exe	488	Load Image	C:\WINDOWS\WinSxS\x86_Microsoft.Wi...	SUCCESS	Image Base: 0x773d0000, Image Size: 0x102000

Showing 66 of 110 470 events (0.0%)

Figure 13: Process related extract from the ProcessMonitor log-file.

Figure 13 shows an extract of the process related events that took place once the file downloaded from Kazaa was double-clicked. The ICQ Lite process created and started the system32 process (1), and then shut itself down (2). (3) shows some of the libraries loaded into the system32 process. Several of the libraries loaded, contained APIs to network-related functions.

- ntdll.dll	- **shell32.dll**	- **wininet.dll**	- **ws2help.dll**	- version.dll
- kernel32.dll	- gdi32.dll	- crypt32.dll	- **icmp.dll**	- **wsock32.dll**
- advapi32.dll	- user32.dll	- msasn1.dll	- **iphlpapi.dll**	- **mswsock.dll**
- **rpcrt4.dll**	- **shlwapi.dll**	- oleaut32.dll	- secur32.dll	- **dnsapi.dll**
- msvcrt.dll	- comctl32.dll	- **ws2_32.dll**	- urlmon.dll	- winrnr.dll
				- **wldap32.dll**

Table 3: Full list of libraries loaded into the system32.exe process. Bold indicates libraries that provide networking functions.

Table 3 shows the full list of libraries loaded into the suspicious process' memory. Several of them, displayed in bold text, provide functions necessary to establish network connections. Information on the libraries was found using [Web43].

The circumstantial evidence found so far strongly indicated the ICQ Lite (new).exe file downloaded from Kazaa was a Trojan horse, most probably designed to communicate/spread via networks.

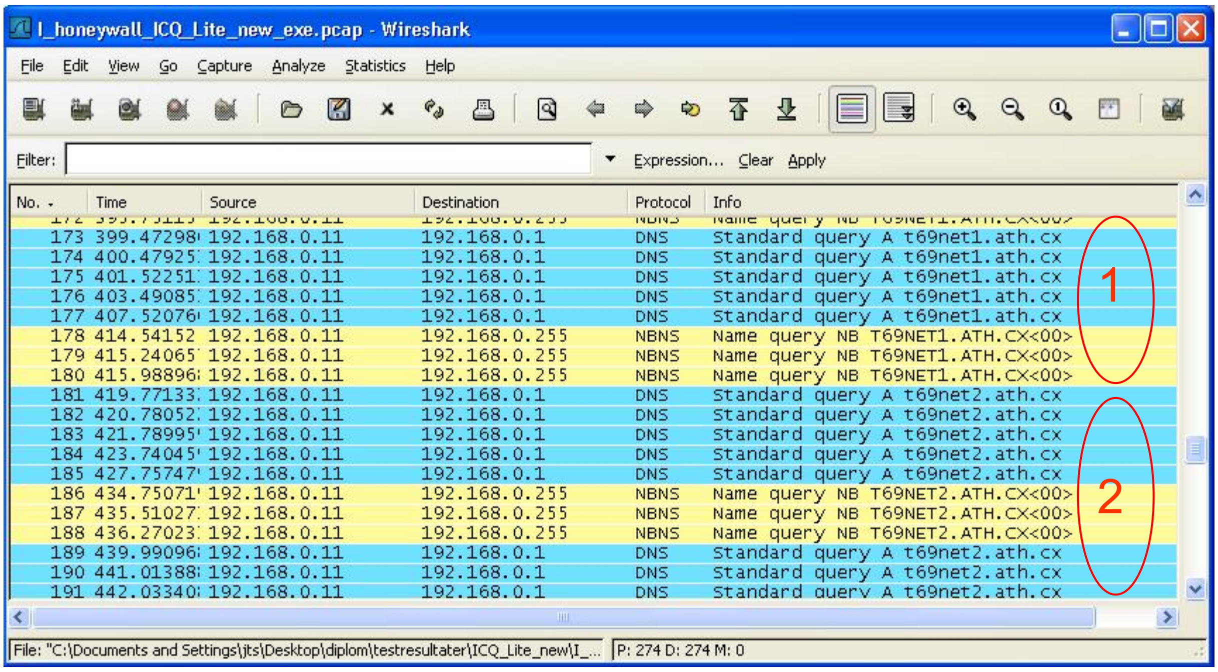
4.2.2. Network-Based Behavioural Analysis

To detect network related behaviour, the *Honeywall Roo* was used in concert with the remote vulnerability scanners *Nmap* and *Nessus*. As presented earlier, the Honeywall logs every packet sent on the network, in addition to supervise network connections with the intrusion detection system *Snort*. *Wireshark* was used to display the packet dump from *Honeywall Roo*.

4.2.2.1.ACR, IDS, Honeypots – Honeywall Roo

At the end of section 4.2.1, it was strongly suggested the ICQ Lite application installed was a Trojan horse, most probably able to communicate/spread via networks. The *Honeywall Roo* was included in the laboratory environment to support network-based detection of hidden functionality, according to principles described by Szor [6] on access control records, Mukherjee et al [31] on intrusion detection systems and Spitzner [8] on honeypots.

Detection of unwanted functionality through network traffic observation could turn out helpful in determining whether the running process, system32.exe, really was a malicious Trojan horse or not.


I_honeywall_ICQ_Lite_new_exe.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
1
2

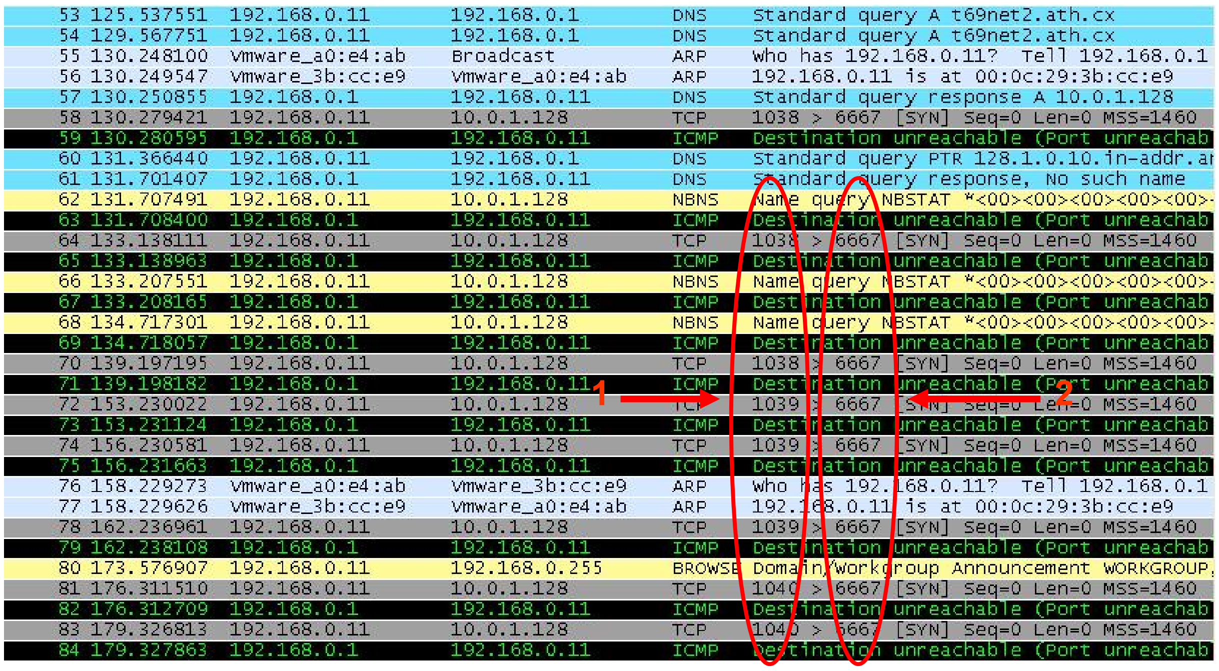
```
No. -  Time       Source         Destination     Protocol  Info
  173  399.47298  192.168.0.11   192.168.0.1     DNS       Standard query A t69net1.ath.cx
  174  400.47925  192.168.0.11   192.168.0.1     DNS       Standard query A t69net1.ath.cx
  175  401.52251  192.168.0.11   192.168.0.1     DNS       Standard query A t69net1.ath.cx
  176  403.49085  192.168.0.11   192.168.0.1     DNS       Standard query A t69net1.ath.cx
  177  407.52076  192.168.0.11   192.168.0.1     DNS       Standard query A t69net1.ath.cx
  178  414.54152  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET1.ATH.CX<00>
  179  415.24065  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET1.ATH.CX<00>
  180  415.98896  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET1.ATH.CX<00>
  181  419.77133  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  182  420.78052  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  183  421.78995  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  184  423.74045  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  185  427.75747  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  186  434.75071  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET2.ATH.CX<00>
  187  435.51027  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET2.ATH.CX<00>
  188  436.27023  192.168.0.11   192.168.0.255   NBNS      Name query NB T69NET2.ATH.CX<00>
  189  439.99096  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  190  441.01388  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx
  191  442.03340  192.168.0.11   192.168.0.1     DNS       Standard query A t69net2.ath.cx

File: "C:\Documents and Settings\jts\Desktop\diplom\testresultater\ICQ_Lite_new\I_...   P: 274 D: 274 M: 0
```

Figure 14: Extract of the traffic dump collected through the Honeywall.

Section 2.3.2.4 explained that an important bot characteristic was the botnet server localization. Figure 14 shows an extract of the traffic dump collected by the Honewall. *Wireshark*, used to display the logged traffic, highlights traffic using different communication protocols in various colours. After activation of the Trojan horse on Bond, DNS queries (blue) and NBNS[8] (yellow) queries were made. The installed Trojan obviously tried to contact its home server(s). As Figure 14 shows, two different servers were tried contacted; t69net1.ath.cx (1) and t69net2.ath.cx (2).

As mentioned earlier, each experiment was run twice. The first time without any Internet access, and the second time with DNS enabled.

[8] NetBIOS Name Servise is a Microsoft proprietary service converting IP addresses to human readable names.



```
53 125.537551 192.168.0.11      192.168.0.1       DNS    Standard query A t69net2.ath.cx
54 129.567751 192.168.0.11      192.168.0.1       DNS    Standard query A t69net2.ath.cx
55 130.248100 Vmware_a0:e4:ab   Broadcast         ARP    Who has 192.168.0.11?  Tell 192.168.0.1
56 130.249547 Vmware_3b:cc:e9   Vmware_a0:e4:ab   ARP    192.168.0.11 is at 00:0c:29:3b:cc:e9
57 130.250855 192.168.0.1       192.168.0.11      DNS    Standard query response A 10.0.1.128
58 130.279421 192.168.0.11      10.0.1.128        TCP    1038 > 6667 [SYN] Seq=0 Len=0 MSS=1460
59 130.280595 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
60 131.366440 192.168.0.11      192.168.0.1       DNS    Standard query PTR 128.1.0.10.in-addr.ar
61 131.701407 192.168.0.1       192.168.0.11      DNS    Standard query response, No such name
62 131.707491 192.168.0.11      10.0.1.128        NBNS   Name query NBSTAT *<00><00><00><00><00>-
63 131.708400 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
64 133.138111 192.168.0.11      10.0.1.128        TCP    1038 > 6667 [SYN] Seq=0 Len=0 MSS=1460
65 133.138963 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
66 133.207551 192.168.0.11      10.0.1.128        NBNS   Name query NBSTAT *<00><00><00><00><00>-
67 133.208165 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
68 134.717301 192.168.0.11      10.0.1.128        NBNS   Name query NBSTAT *<00><00><00><00><00>-
69 134.718057 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
70 139.197195 192.168.0.11      10.0.1.128        TCP    1038 > 6667 [SYN] Seq=0 Len=0 MSS=1460
71 139.198182 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
72 153.230022 192.168.0.11      10.0.1.128        TCP    1039 > 6667 [SYN] Seq=0 Len=0 MSS=1460
73 153.231124 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
74 156.230581 192.168.0.11      10.0.1.128        TCP    1039 > 6667 [SYN] Seq=0 Len=0 MSS=1460
75 156.231663 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
76 158.229273 Vmware_a0:e4:ab   Vmware_3b:cc:e9   ARP    Who has 192.168.0.11?  Tell 192.168.0.1
77 158.229626 Vmware_3b:cc:e9   Vmware_a0:e4:ab   ARP    192.168.0.11 is at 00:0c:29:3b:cc:e9
78 162.236961 192.168.0.11      10.0.1.128        TCP    1039 > 6667 [SYN] Seq=0 Len=0 MSS=1460
79 162.238108 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
80 173.576907 192.168.0.11      192.168.0.255     BROWSE Domain/Workgroup Announcement WORKGROUP,
81 176.311510 192.168.0.11      10.0.1.128        TCP    1040 > 6667 [SYN] Seq=0 Len=0 MSS=1460
82 176.312709 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
83 179.326813 192.168.0.11      10.0.1.128        TCP    1040 > 6667 [SYN] Seq=0 Len=0 MSS=1460
84 179.327863 192.168.0.1       192.168.0.11      ICMP   Destination unreachable (Port unreachab
```

Figure 15: Extract of the traffic dump collected through the Honeywall immediately after DNS was activated.

Figure 15 illustrates what happened immediately after access to the DNS server was allowed. Once the DNS queries resolved the server name to an IP address, connection attempts were tried towards the IP address. There are two especially interesting observations shown in the picture. The first arrow points towards the source ports used during the connection attempts. After two connection attempts from each source port, the value of the port number was incremented by one. The reason for this is most likely that the Trojan horse tried to evade the firewall that was blocking the connection attempts. By trying different source ports it could, in the real world, eventually hit an open port.

The second arrow is pointed towards the destination port of the connection attempts. Port 6667 on the remote host was tried contacted. Port 6667 is commonly used by IRC networks. The home server localization procedure, followed by connection attempts using a common IRC port, could be indicating a bot was installed.

4.2.2.2.Remote Vulnerability Checking – Nmap and Nessus

To further investigate the victim, remote vulnerability checking was performed. Skaggs et al [34] suggested vulnerability checking to look for backdoors and the like. These checks could also be used to determine whether existing vulnerabilities on the victim had been patched or if new vulnerabilities were introduced.

In the first experiment run, without DNS access, *Nmap* was used to quickly determine the port status of the victim. An *Nmap* scan made before the ICQ Lite application was introduced, reported the status to be as shown in Figure 16. Three ports, 135, 139 and 445, were open.

```
Not shown: 65532 closed ports
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
```

Figure 16: Nmap scan result of a clean system

The *Nmap* scan made after the suspicious system32 process had started running, gave the same result. To verify this, a *Nessus* scan was performed and the results from this utility before and after installation of ICQ Lite were compared. Neither *Nessus* nor *Nmap* could find any changes on the victim host.

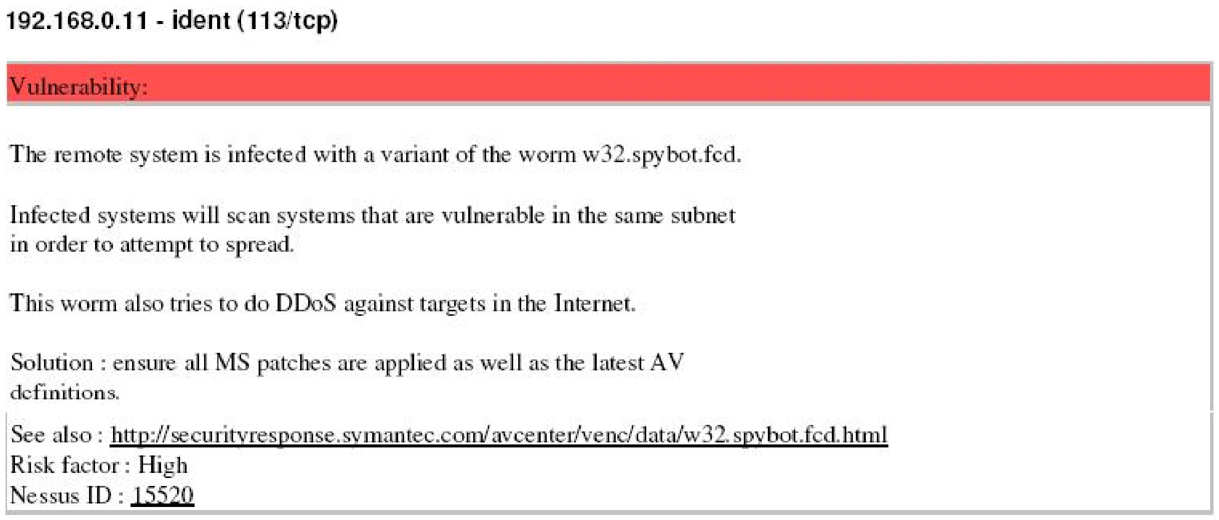
In the second experiment run, things started happening when the victim successfully performed a DNS query. An *Nmap* scan made after the connection attempts illustrated in Figure 15 returned an interesting result.

```
Not shown: 65531 closed ports
PORT    STATE SERVICE      VERSION
113/tcp open  auth?
135/tcp open  msrpc        Microsoft Windows RPC
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds Microsoft Windows XP microsoft-ds
```

Figure 17: Nmap scan of compromised system, after successful DNS queries.

Figure 17 illustrates the results returned by the *Nmap* scan performed after DNS access was given. In addition to the three open ports found on a clean system, port 113 was reported open. RFC 1700 [35] reserves this port to authentication services. The introduction of a new opened listening port to the system looked very suspicious. With all the information collected earlier in mind, the most likely explanation to this new port was that a backdoor had been installed on the system.

Nessus was run to se if any of the open ports responded to attacker like behaviour.



192.168.0.11 - ident (113/tcp)

Vulnerability:

The remote system is infected with a variant of the worm w32.spybot.fcd.

Infected systems will scan systems that are vulnerable in the same subnet
in order to attempt to spread.

This worm also tries to do DDoS against targets in the Internet.

Solution : ensure all MS patches are applied as well as the latest AV
definitions.

See also : http://securityresponse.symantec.com/avcenter/venc/data/w32.spybot.fcd.html
Risk factor : High
Nessus ID : 15520

Figure 18: Extract from the Nessus scan report created after DNS access was given.

Figure 18 shows an extract of the scan report generated by *Nessus*. One of *Nessus*' plug-in modules recognized the response of port 113 during the scan. The port was opened as a backdoor, and *Nessus* strongly suggested a bot variant was responsible.

By looking at all the evidence collected so far, this scan report added the final evidence for believing the ICQ Lite (new).exe really was a Trojan horse hiding bot functionality.

4.2.3. Host-Based Behavioural Analysis - Revisited

ProcessExplorer was the first application used in this test to discover suspicious activity. This utility has another nifty function in addition to what is presented so far; it has the ability to read strings from each process, both from the process image on the hard disk and from the process running in memory. Even though this subject may border to static analysis techniques, it is well worth mentioning.

String dump from image	String dump from memory
eOM.	password accepted.
gwK	*failed host auth by %s(%s).
mto	NOTICE %s :host auth failed.
dr~j	NOTICE %s :
Teg_	PING %s
c^&sH	PING
mAX	NOTICE %s :
J#ya	VERSION %s
KPA	VERSION
pyLi4	*failed pass auth by %s(%s).
&Cy6u	NOTICE %s :your attempt has been logged.
mkO2K0	NOTICE %s :pass auth failed.
pDH{x	hook
q(_HV	*failed cmd exec by %s(%s) with cmd %s.
YCZ	KICK %s %s :%s
TRA	MODE %s +b *!%s
#Jjg	NOTICE
-ipx	PRIVMSG
Jso	joined channel %s.

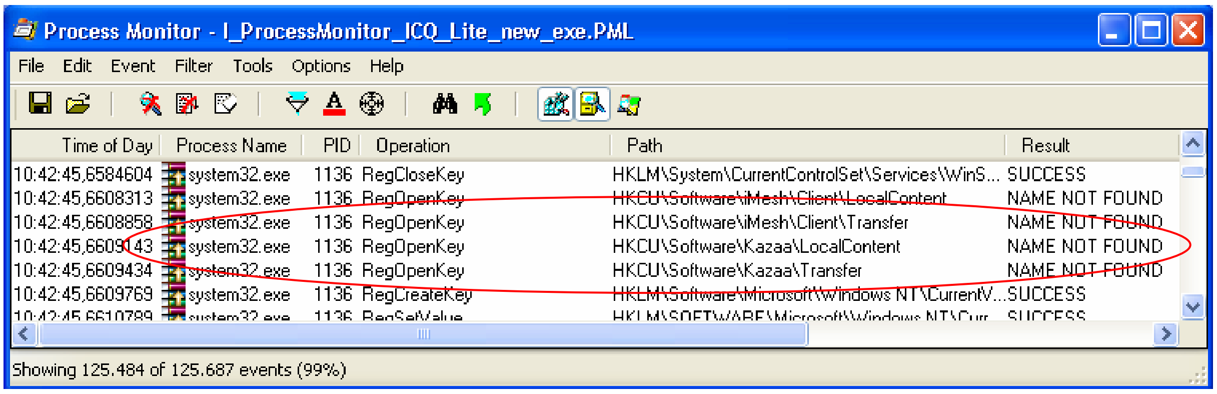
Table 4: Extract from string dump of a process' image file and from a process running in memory[9].

Table 4 illustrates the usefulness of reading strings from processes running in memory. To the left is an extract of the string dump from the System32.exe image file. The result is consistent with the fact that the process is reported to be packed. Only a limited amount of information can be collected from the packed application body. Processes running in memory, however, are de-packed/decrypted to their original content. Dumping strings from processes running in memory may therefore lead to important information, as displayed in the right column in Table 4. Section 2.3.2.1 described botnet control mechanisms, where words like NOTICE and PRIVMSG, among others, are defined by the IRC protocol and therefore also related to control of IRC bots.

The string dump from the running process concluded all suspicions towards the ICQ Lite application. By downloading and installing what looked like a benign application, the victim computer became infected with a malicious Trojan horse concealing an IRC bot.

[9] The table columns are not correlated. They are randomly extracted from the string dumps.

Event though the limitation set in section 1.4 limits the analysis of registry activity to looking at the insertion of registry keys, curiosity led to an important discovery that should be mentioned. When studying the ICQ Lite application the following observation was made:


Process Monitor - I_ProcessMonitor_ICQ_Lite_new_exe.PML

```
Time of Day       Process Name  PID   Operation      Path                                            Result
10:42:45,6584604  system32.exe  1136  RegCloseKey    HKLM\System\CurrentControlSet\Services\WinS...  SUCCESS
10:42:45,6608313  system32.exe  1136  RegOpenKey     HKCU\Software\iMesh\Client\LocalContent         NAME NOT FOUND
10:42:45,6608858  system32.exe  1136  RegOpenKey     HKCU\Software\iMesh\Client\Transfer             NAME NOT FOUND
10:42:45,6609143  system32.exe  1136  RegOpenKey     HKCU\Software\Kazaa\LocalContent                NAME NOT FOUND
10:42:45,6609434  system32.exe  1136  RegOpenKey     HKCU\Software\Kazaa\Transfer                    NAME NOT FOUND
10:42:45,6609769  system32.exe  1136  RegCreateKey   HKLM\Software\Microsoft\Windows NT\CurrentV...  SUCCESS
10:42:45,6610789  system32.exe  1136  RegSetValue    HKLM\SOFTWARE\Microsoft\Windows NT\Curr...      SUCCESS

Showing 125.484 of 125.687 events (99%)
```

Figure 19: Malicious process looking for installed applications.

Figure 19 shows how the malicious system32 process, created by ICQ Lite (new).exe, tries to open registry keys belonging to the Kazaa and iMesh[10] applications. As the figure shows, these registry keys were not found on the test system. Clearly, this query had to mean something. To do a further investigation of the Trojan application, Kazaa was installed on a clean system. Next, the ICQ Lite application was installed, and the same tests as presented above were performed.

The result was quite interesting. The *ProcessMonitor* log file revealed that an extensive list of new files had been created on the system.

Battlefield1942_keygen.exe	Winamp 3.8.exe	GTA 3 Crack.exe	ACDSee 5.5.exe
WinZip 9.0b.exe	MediaPlayer Update.exe	GTA 3 patch (no cd).exe	DivX Video Bundle 6.5.exe
SmartFTP 2.0.0.exe	UT2003_patch.exe	Hitman_2_no_cd_crack.exe	Global DiVX Player 3.0.exe
ICQ Lite (new).exe	KaZaA Hack 2.5.0.exe	Mafia_crack.exe	QuickTime_Pro_Crack.exe
ICQ Pro 2003b (new beta).exe	DirectDVD 5.0.exe	Neverwinter_Nights_licence.exe	KaZaA Lite (New).exe
ICQ Pro 2003a.exe	Flash MX crack (trial).exe	NHL 2003 crack.exe	iMesh 3.7b (beta).exe
AOL Instant Messenger.exe	Ad-aware 6.5.exe	WarCraft_3_crack.exe	iMesh 3.6.exe

Table 5: Extract of files created by the system32 process.

[10] iMesh is a file-sharing utility similar to Kazaa.

Table 5 shows an extract of the executable files generated once the ICQ Lite application was installed on a system with the Kazaa application present. A total of 94 different files were generated and placed in a folder that was shared to the Kazaa network.

This illustrates that the behaviour observed during installation is dependant on the environment applications are installed onto.

4.2.4. Test Results – Summary

The following section will try to summarize the most important findings from each test. The results are obtained in accordance with the data collected through the thorough example given above. Emphasize is given to the behaviour of the HackerDefender rootkit as this was not presented in the previous section. Due to the size and amount collected experiment data, the log files generated during each experiment can be found on a DVD appended to this report.

4.2.4.1.Event Monitoring – ProcessExplorer

The following table illustrates what was observed with *ProcessExplorer* after each application was activated.

Application	Suspicious process found	Running process
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	Yes	system32.exe - packed
ICQ hacks.exe Winzip_Crack.exe	Yes	ms.bak.temp.exe - packed
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	Yes	system32.exe - packed
Internat.exe	Yes	internat.exe - packed
Winlist.exe	Yes	wvvqvte.exe - packed
Msq32.exe	Yes	msq32.exe - packed
Via.exe	Yes	adirka.exe - packed
Bot.exe	Yes	okkczbk.exe - packed
HackerDefender rootkit	No	

Table 6: Suspicious processes observed with *ProcessExplorer*.

From Table 6 one can se the processes observed running after activation of the different applications. Each process was highlighted purple by *ProcessExplorer*, which means the process image was packed. Packing is a common obfuscation technique used by malware, as described in section 2.3.1.2.

As previously explained, *ProcessExplorer* is also capable of reading strings from processes' image files and from processes running in memory. In every case, strings collected from the de-packed process running in memory returned useful information about the functionality. Table 4 gave an example of what information can be found in the de-packed process.

ProcessExplorer depends on the Windows API to acquire the list of running processes. HackerDefender could not be observed by this utility as the rootkit controls the functions used to list running processes.

4.2.4.2.Auto-Start Extensibility Points - AutoRuns

The table below illustrates which changes in auto-start extensibility points were observed using *AutoRuns*.

Application	Changes found	Auto-start extensibility point used
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Winlogon\Shell Image path: C:\WINDOWS\system32\System32.exe
ICQ hacks.exe Winzip_Crack.exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\RDLL Image path: File not found: RunDl116.exe
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Winlogon\Shell Image path: C:\WINDOWS\system32\System32.exe
Internat.exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\Internat.exe
Winlist.exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\ixgqmjj.exe HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\ixgqmjj.exe
Msq32.exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\msq32.exe HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\msq32.exe
Via.exe	Yes	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\adirka.exe
Bot.exe	Yes	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ Image path: C:\windows\system32\idkhvxs.exe
HackerDefender rootkit	No	

Table 7: Auto-start extensibility points found using *AutoRuns*.

As Table 7 illustrates, each of the malware specimens tested, utilized auto-start extensibility points to be able to survive reboots of the compromized computer, seemingly except HackerDefender. HackerDefender also hide its registry entries, in addition to hiding running processes as seen with *ProcessExplorer*.

4.2.4.3.Cross-Time Diff – Osiris

File integrity checks, according to the cross-time diff principles also returned interesting results.

Application	**File changes observed**	**New files**
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	Yes	C:\Documents and Settings\bond \Desktop\ICQ Lite (new).exe C:\WINDOWS\system32\system32.exe
ICQ hacks.exe Winzip_Crack.exe	Yes	C:\Documents and Settings\bond \Desktop\ICQ hacks.exe C:\WINDOWS\System32\ms_32.exe C:\WINDOWS\System32\ms_bak.tmp.exe
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	Yes	C:\Documents and Settings\bond\Desktop\ICQ Lite beta (b2253).exe C:\WINDOWS\system32\System32.exe C:\WINOWS\system32\xms32.exe
Internat.exe	Yes	C:\WINDOWS\System32\Internat.exe
Winlist.exe	Yes	C:\WINDOWS\System32\ ixgqmjj.exe
Msq32.exe	Yes	C:\WINDOWS\System32\msq32.exe
Via.exe	Yes	C:\Documents and Settings\bond \Desktop\via.exe C:\WINDOWS\system32\adirka.dll C:\WINDOWS\system32\adirka.exe C:\WINDOWS\system32\zlbw.dll
Bot.exe	Yes	C:\WINDOWS\system32\oqbxtdd.exe
HackerDefender	No	

Table 8: File changes found using *Osiris*.

Table 8 shows the file changes observed after each application was activated. Every application, except HackerDefender, created new files in the C:\WINDOWS\System32 folder. This is the folder where several legitimate Windows applications and libraries are stored. Because of the number of files in this directory, it is a well considered place to hide malware. Internat.exe, winlist.exe, msq32.exe and bot.exe deleted their installation files during execution, thus no desktop files were reported for these applications.

No changes to existing files were observed, thus it is not likely that virus infection techniques were utilized by these applications.

Even malware detection through cross-time diff principles was evaded by the HackerDefender rootkit. So far, HackerDefender had slipped away detection from all applications utilizing the MS Windows API to collect data.

4.2.4.4.Event Monitoring – ProcessMonitor

ProcessMonitor logged event data on process- , registry- and file activity from before applications were activated and until five minutes after, as explained earlier. By studying the log files generated by this utility, creation of files, auto-start registry entries and the creation of processes for each application were found. All results presented in Table 6, Table 7 and Table 8 were also observed by studying the *ProcessMonitor* log file.

Even more valuable information could be extracted with this utility. When studying which registry entries were set, on could see that *AutoRuns* did not give the full picture. The registry settings presented in table 8 were not displayed by *AutoRuns*:

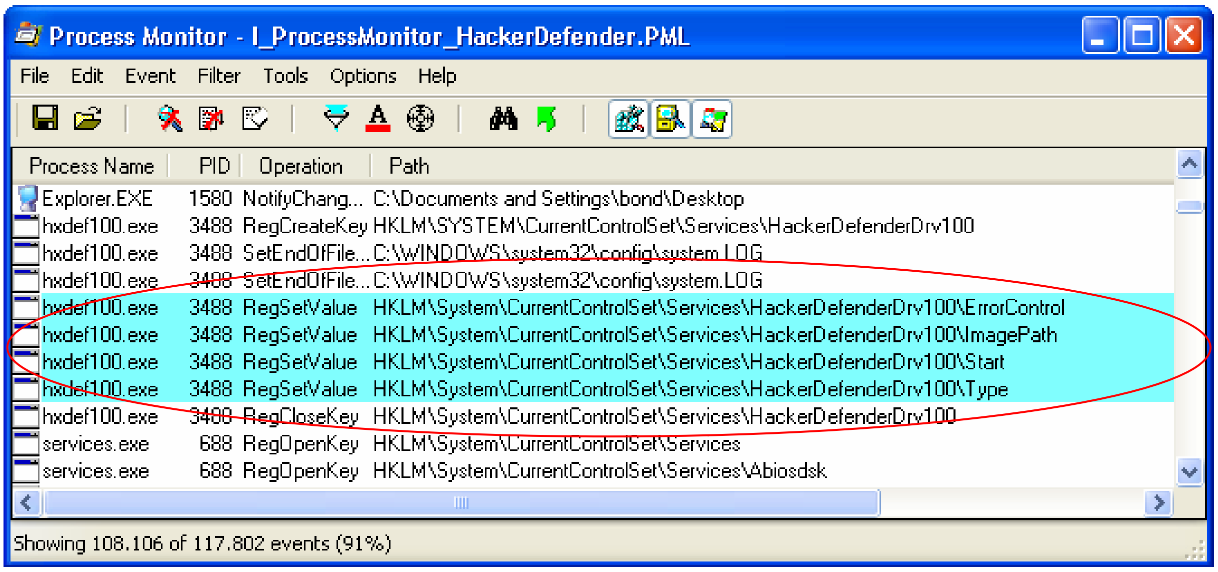
Application	**Registry entry**
ICQ hacks.exe Winzip_Crack.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices Image path: File not found: RunDll16.exe
Internat.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices Image path: C:\Windows\system32\Internat.exe HKCU\Software\Microsoft\OLE Image path: C:\Windows\system32\Internat.exe
Msq32.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices Image path: C:\Windows\system32\msq32.exe
Bot.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices Image path: C:\Windows\system32\idkhvxs.exe HKCU\Software\Microsoft\OLE Image path: C:\Windows\system32\ idkhvxs.exe

Table 9: Auto-start registry entries not observed by AutoRuns

By doing a Google search with the query "HKCU\Software\Microsoft\OLE", which was a registry setting inserted by Internat.exe and Bot.exe, an overwhelming majority of the results contained information on Trojans and bots. Microsoft [Web48] reports the RunServices entry to contain auto-starting entries. However, the path is only operational on the Windows 95, Windows 98 and Windows Me platforms. This indicated that the malware specimens presented in table 8, most probably are backwards compatible with older operating system platforms. This may also be the explanation to why these settings were not reported by *AutoRuns*.

In addition to the information, one could read which libraries were loaded into the malicious processes' memory. Appendix D lists all libraries loaded into the different processes. Every applications loaded libraries containing networking functions, which are essential for malware containing bot and backdoor functionality to function.

What is especially interesting with the event monitoring principle is that the rootkit HackerDefender could not evade detection. Rootkits are not able to hide their content before they are properly installed and active.


Process Monitor - I_ProcessMonitor_HackerDefender.PML
File Edit Event Filter Tools Options Help

```
Process Name   PID  Operation      Path
Explorer.EXE   1580 NotifyChang... C:\Documents and Settings\bond\Desktop
hxdef100.exe   3488 RegCreateKey   HKLM\SYSTEM\CurrentControlSet\Services\HackerDefenderDrv100
hxdef100.exe   3488 SetEndOfFile...C:\WINDOWS\system32\config\system.LOG
hxdef100.exe   3488 SetEndOfFile...C:\WINDOWS\system32\config\system.LOG
hxdef100.exe   3488 RegSetValue    HKLM\System\CurrentControlSet\Services\HackerDefenderDrv100\ErrorControl
hxdef100.exe   3488 RegSetValue    HKLM\System\CurrentControlSet\Services\HackerDefenderDrv100\ImagePath
hxdef100.exe   3488 RegSetValue    HKLM\System\CurrentControlSet\Services\HackerDefenderDrv100\Start
hxdef100.exe   3488 RegSetValue    HKLM\System\CurrentControlSet\Services\HackerDefenderDrv100\Type
hxdef100.exe   3488 RegCloseKey    HKLM\System\CurrentControlSet\Services\HackerDefenderDrv100
services.exe    688 RegOpenKey     HKLM\System\CurrentControlSet\Services
services.exe    688 RegOpenKey     HKLM\System\CurrentControlSet\Services\Abiosdsk
Showing 108.106 of 117.802 events (91%)
```

Figure 20: Auto-start entries inserted by HackerDefender, observed through *ProcessMonitor*.

Figure 20 illustrates that also rootkits utilize auto-start extensibility points to be able to remain active on compromised hosts, even after reboots. HackerDefender inserted itself as a service that would start every time the computer was powered on. By a further study of the *ProcessMonitor* log file, one could see that the rootkit even inserted two more auto-start entries; one that would activate HackerDefender if the user logged into the compromised Windows operating system in minimal safe-mode, and on that would activate the rootkit if safe-mode with networking was used.

AutoRuns, *Osiris* and, to a certain extent, *ProcessExplorer* were used to check for changes after the rootkit had become active. HackerDefender was thus able to evade detection by these tools according to the principles described in section 2.4.1.1. The fact that suspicious events, like the insertion of auto-starting registry entries, were detected by *ProcessMonitor* illustrates the power of real time event monitoring as a detection principle.

4.2.4.5.Cross-View Diff / Rootkit Detection

Cross-view diff was introduced as a principle to detect software with stealth properties. The HackerDefender rootkit have so far demonstrated the effectiveness of the stealth category of malware. If *ProcessMonitor* had not explicitly been configured to record events during installation, nothing would have suggested the installation of anything suspicious with the tools presented so far.

Application	**Suspicious activity found**	**Discrepancies found**
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	No	
ICQ hacks.exe Winzip_Crack.exe	No	
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	No	
Internat.exe	No	
Winlist.exe	No	
Msq32.exe	No	
Via.exe	No	
Bot.exe	No	
HackerDefender rootkit	Yes	Examples of registry entries hidden from the Windows API: HKLM\SYSTEM\ControlSet001\Services\HackerDefender100 HKLM\SYSTEM\ControlSet001\Services\HackerDefenderDrv100 Examples of files hidden from the Windows API: C:\Documents and Settings\bond\Desktop\hxdef100\hxdef100.exe C:\Documents and Settings\bond\Desktop\hxdef100\hxdef100.ini C:\Documents and Settings\bond\Desktop\hxdef100\hxdefdrv.sys

Table 10: Results of system scans using *RootkitRevealer*.

Table 10 illustrates that cross-view diff principles are possible to utilize for detection of rootkits. Both hidden files and registry settings were found using *RootkitRevealer*. What was interesting to observe was that RootkitRevealer did not report the registry entries presented in Figure 20. Instead registry settings, like the two presented in Table 10, were reported. *ProcessMonitor* reported registry settings in the path:

HKLM\SYSTEM\CurrentControlSet\Services\

while *RootkitRevealer* discovered hidden entries in the path:

HKLM\SYSTEM\ControlSet001\Services\

The discrepancies between these two observations may indicate that HackerDefender was able to hide its activity from *ProcessMonitor* once it became fully active. The registry seemed to have been edited unnoticed by the event monitor. In addition, HackerDefender seem to be able to hide some of its content, like the registry entries presented in Figure 20, even to *RootkitRevealer*.

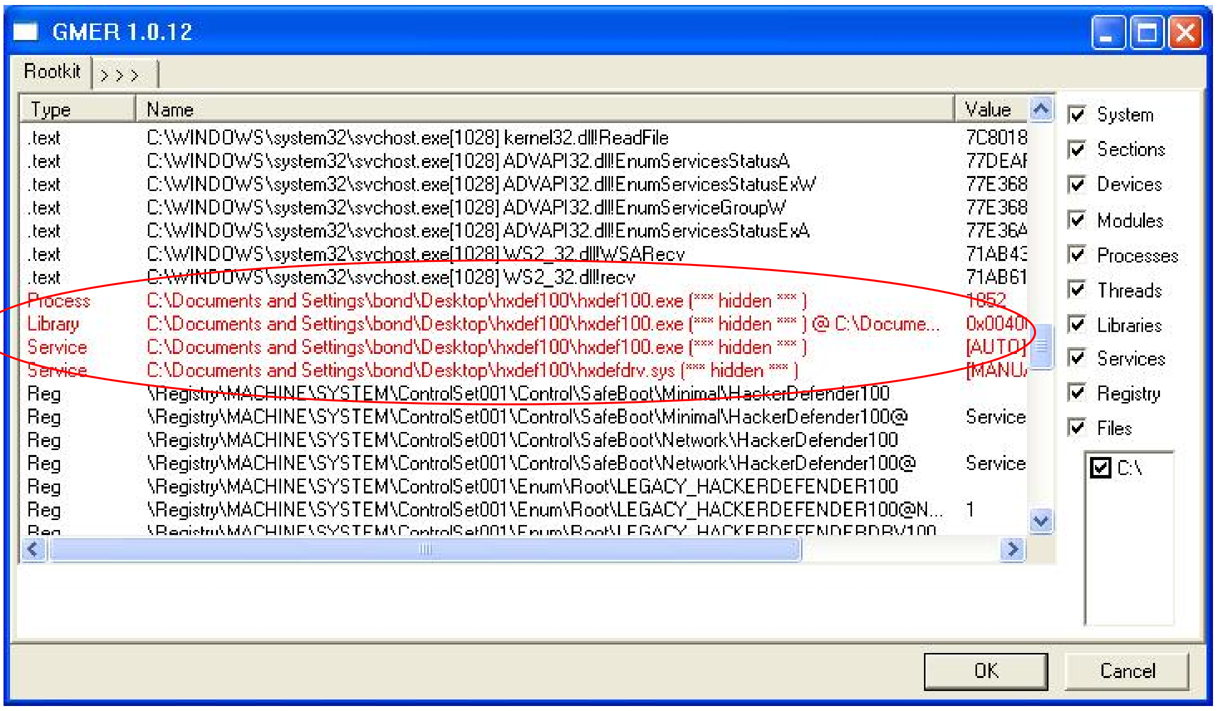
Gmer performs more extensive tests than *RootkitRevealer*. In addition to cross-view diff principles API hooks are discovered with this utility.

\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100
\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100
\Registry\MACHINE\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDER100
\Registry\MACHINE\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDERDRV100
\Registry\MACHINE\SYSTEM\ControlSet001\Services\HackerDefender100
\Registry\MACHINE\SYSTEM\ControlSet001\Services\HackerDefenderDrv100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\HackerDefender100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Control\SafeBoot\Network\HackerDefender100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_HACKERDEFENDER100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_HACKERDEFENDERDRV100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Services\HackerDefender100
\Registry\MACHINE\SYSTEM\CurrentControlSet\Services\HackerDefenderDrv100

Table 11: Hidden registry paths discovered by *Gmer*

Table 11 shows the registry paths reported by *Gmer* as hidden. The table shows that the registry entries discovered by both *ProcessMonitor* and *RootkitRevealer* all were valid.

An important property of Gmer over RootkitRevealer is that this utility also searches for hidden processes, libraries and services, not only hidden files and folders.



Type	Name	Value
.text	C:\WINDOWS\system32\svchost.exe[1028] kernel32.dll!ReadFile	7C8018
.text	C:\WINDOWS\system32\svchost.exe[1028] ADVAPI32.dll!EnumServicesStatusA	77DEAF
.text	C:\WINDOWS\system32\svchost.exe[1028] ADVAPI32.dll!EnumServicesStatusExW	77E368
.text	C:\WINDOWS\system32\svchost.exe[1028] ADVAPI32.dll!EnumServiceGroupW	77E368
.text	C:\WINDOWS\system32\svchost.exe[1028] ADVAPI32.dll!EnumServicesStatusExA	77E36A
.text	C:\WINDOWS\system32\svchost.exe[1028] WS2_32.dll!WSARecv	71AB43
.text	C:\WINDOWS\system32\svchost.exe[1028] WS2_32.dll!recv	71AB61
Process	C:\Documents and Settings\bond\Desktop\hxdef100\hxdef100.exe (*** hidden ***)	1852
Library	C:\Documents and Settings\bond\Desktop\hxdef100\hxdef100.exe (*** hidden ***) @ C:\Docume...	0x0040
Service	C:\Documents and Settings\bond\Desktop\hxdef100\hxdef100.exe (*** hidden ***)	[AUTO]
Service	C:\Documents and Settings\bond\Desktop\hxdef100\hxdefdrv.sys (*** hidden ***)	[MANU
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100	
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100@	Service
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100	
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100@	Service
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDER100	
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDER100@N...	1
Reg	\Registry\MACHINE\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDERDRV100	

Figure 21: Hidden process, library and services discovered by *Gmer*.

Figure 21 shows process, library and services hidden by HackerDefender, discovered by Gmer.

4.2.4.6.Network Activity

Network activity was recorded for each utility tested. The following table summarizes the home-server localization procedures used by the malware.

Application	Home Server Localization	Method	Home server	Destination port
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	Yes	DNS and NBNS queries	t69net1.ath.cx t69net2.ath.cx	6667
ICQ hacks.exe Winzip_Crack.exe	Yes	DNS and NBNS queries	Xhum.ath.cx eduBox.ath.cx	6667
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	Yes	DNS and NBNS queries	rbase01.ath.cx rb08r15.ath.cx	6667
Internat.exe	Yes	DNS queries	Fareed.arab-hacker.org	6667
Winlist.exe	Yes	DNS queries	luani.chatoni.com	6667
Msq32.exe	Yes	DNS and NBNS queries	mx.albadal.net	7777
Via.exe	Yes	Hard coded list of IP addresses		80
Bot.exe	Yes	Hard coded IP address		7000
HackerDefender rootkit	No			

Table 12: Overview of home-server localization procedures.

As Table 12 presents, each of the malware specimens tested had a form of home-server localization procedure. Since no internet connection was provided, except for access to a local DNS server, it was easy to discover the traffic generated by the newly installed applications. At a regular basis, all of them tried contact their home command and control servers. These regular traffic patterns were easy to discover by a manual inspection of the network traffic dump file collected by *Wireshark* and the *Honeywall*. Every application was observed to increment the value of the source port number after failed connection attempts.

As a consequence of each test being performed without internet connection, no alerts were raised by the IDS *Snort* or other logging mechanisms of the *Honeywall*.

HackerDefender is not designed to transmit data automatically. Every connection to the backdoors created by this rootkit has to be initiated by a remote user. Communication with HackerDefender was not tested.

4.2.4.7.Remote Vulnerability Scanning

As seen in the ICQ Lite example, remote vulnerability scanning returned valuable results in several of the tests.

Application	**Changes before DNS**	**Changes after DNS**	**Nmap observation**	**Nessus observations**
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	No	Yes	Port 113 opened	Port 113 opened by w32.spybot.fcd
ICQ hacks.exe	No	Yes	Port 113 opened	Port 113 opened, requests are forwarded to port 6667. Probably caused by an IRC bot
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	No	Yes	Port 113 opened	Port 113 opened, requests are forwarded to port 6667. Probably caused by an IRC bot
Internat.exe	No	No		
Winlist.exe	Yes	Yes	Port 113 opened	Port 113 opened. Port responds to empty queries with a random user ID. Behaviour indicative of an IRC bot.
Msq32.exe	No	Yes	Port 113 opened	Port 113 opened. Port responds to empty queries with a random user ID. Behaviour indicative of an IRC bot.
Via.exe	No	No		
Bot.exe	No	No		
HackerDefender rootkit	No	No		

Table 13: Summary of remote vulnerability scan results by use of *Nessus* and *Nmap*

As seen in Table 13, remote vulnerability scanning can be useful when analyzing software behaviour. Several of the applications opened backdoors on the compromised computer, which were discovered with the remote tools.

HackerDefender was not reported to cause any changes to its compromised host. As described earlier, this rootkit opens backdoors on its victims. Since the backdoors are created by hooking into already open communication ports, no changes are observed through remote port scanners like *Nmap*. *Nessus*, however, has a plug-in module specially designed for

HackerDefender detection. Still, it was not capable of discovering the rootkit backdoors. The HackerDefender plug-in module was written in 2004, and the rootkit may have evolved since then. It is not further investigated why *Nessus* could not discover HackerDefender.

4.2.4.8.Anit-Virus Scan

ClamWin was used at the end of the experiment phase to determine what kind of malware had been tested.

Application name	Malware
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	Worm.P2P.Tanked.11
ICQ hacks.exe Winzip_Crack.exe	Worm.SdDrop.A
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	Worm.SdDrop.A
Internat.exe	Trojan.SdBot-4704
msq32.exe	Trojan.SdBot-4967
winlist.exe	Trojan.SdBot-5919
bot.exe	Trojan.IRCBot-445
via.exe	Trojan.Small-1348
hxdef100.zip	Trojan.Hackdef.084-prog

Table 14: ClamWin scan results.

It does not seem to be any correlated naming conventions between different anti-virus providers. Clam AV/*ClamWin* does not provide descriptions of the malware they are capable of detecting. What is interesting with the scan, however, is that every application tested is confirmed to be malware. This confirmed the conclusions made based on the behavioural tests. From the scan result presented in table 14, one can suggest that during the experiment phase, six different malware families were tested.

5. DISCUSSION

This chapter will attempt to determine whether or not the laboratory setup designed, and the method used, is adequate for detecting hidden malicious software functionality through dynamic behavioural analysis. This will be done by comparing properties described in the theory section with the results obtained from the experiments. Detection of bot and rootkit functionality is emphasized.

During the experiments, thirteen different malware specimens were tested. Symantec presented in their Internet security threat report for July to December 2006 [Web1] that during that time period:

- their honeypots captured almost 150 previously unseen malicious code threats
- they received reports of more than 8.000 new variants of Windows malware

From Table 14, one can extract the fact that of the twelve bot specimens analysed, there were only five different bot families. Only one rootkit example was studied, and this was a so called user-mode rootkit. Kernel-mode rootkits were not tested. By looking at the numbers presented by Symantec compared to the number of malware specimens analysed during the experimentation phased, one can understand that it will be difficult and inaccurate to conclude to the generality of the detection abilities for the laboratory environment. Some key observations can, however, be extracted.

5.1. Laboratory Environment

The laboratory environment was designed based on theoretical knowledge gained by looking at the characteristics of bots and rootkits, in relation with a selection of malware detection principles suggested to detect similar characteristics. This theory was presented in section 2.3, 2.4 and 2.5, respectively.

The sources referred to throughout this document were tried kept as recent as possible to make sure the laboratory was best possibly suited to meet current challenges. The following example can be given on how this has influenced on the laboratory design:

> As presented earlier, malware trends observed by Symantec in the period July – December 2006 [Web1], indicates that authors of malicious code tends to use a staged download principle. To be able to analyze all kinds of software, it is therefore not sufficient to design an analysis laboratory solely based on the sandbox principles, which is the recommended and most common approach [6, 7]. Honeypot technologies were therefore included in the laboratory setup to increase the flexibility of the laboratory environment to allow safe, controlled access to internet resources.

There are, however, drawbacks to this approach. A lot of important research has been made on malware detection and analysis in the years before 2000. Important input may have been missed. On the other hand, the botnet problems and the evolution of rootkit technologies are not yet fully documented because of their rapid development the last couple of years. The best solution was therefore found to follow the principle of studying more recent research on the topics.

To increase the flexibility of the laboratory environment, virtual machine technology was utilized. This technology can multiply the effectiveness and the precision of the analysis phase, as virtual machines can be reverted to known clean states with the exact same start conditions for each experiment in seconds. Holz and Raynal [36] studied how different techniques and tools can be used to detect suspicious environments, like virtual machines. By looking at the MAC address of the network card or study specifications of hardware, like the hard drive and video card, attackers can fingerprint virtual environments. Section 2.3.1.3 on bot deception characteristics presented the possibility that bots, and thus also malware in general, may include functionality to detect these virtual surroundings. This may influence on the experiment result:

If any of the applications tested had been designed with the above mentioned fingerprinting functionality, the result might have been that nothing had happened at all after activating the Trojan files. The tests presented in this report were carried out with Trojan horses that only contained functionality of the malicious code itself. However, if the bot functionality had been bundled with a real ICQ application, for instance, the installation of the original application could have proceeded as normal. The malicious code, on the other hand, could have discovered the suspicious surroundings and remained inactive. False negative results are even worse than false positives. Wrong conclusions may lead to the installation of the malicious code in production environments. Since none of the malware specimens tested during this work contained such functionality, it is difficult to determine whether or not the malware's probes to discover virtual environments would have been detected.

Another important aspect regarding virtual machine technology was mentioned in section 3.3. Natvig [4] expressed concerns about malware being able to escape virtual surroundings to infect the host operating system. By using a totally different operating system as host for the virtual victim machine, in addition to the system hardening measures performed in laboratory in general, the setup should be safe from this threat. However, new vulnerabilities and exploit methods are constantly found and developed. The laboratory environment should therefore never be connected to production environments, even though it is supposed to be clean.

There is a solution to the two concerns regarding virtual machine technology, which has better flexibility than formatting and reinstalling each operating system, and the whole analysis environment between each test. When the laboratory is appropriately set up, images of the clean computers can be made and stored in a secure location. After each test, these images could be mirrored back to the laboratory machines and return the environment to a known clean state. This would provide analysis opportunities in native environments. The only drawback to this approach is that it is more time consuming than restoring a virtual environment. Dependant of the technique, the time to restore a computer can vary from approximately twenty minutes to close to an hour, by use of CD recovery and network booting, respectively[11]. Virtual machine technology was therefore found best fit.

The analysis tools used in this laboratory setup was chosen based on how their descriptions corresponded to properties described in the presented malware detection theory. A requirement set in section 1.4 was that every tool should be free of charge. This requirement was set to distinguish the work presented in this report, and the related work presented in section 1.3. In addition, research on the topics covered in this report is important and the possibility to study the subject should be feasible to organizations with low budgets or to e.g. students. A lot of commercial software/malware analysis tools exist. These tools may add

[11] These are rough, undocumented estimations based on previous experience.

extra value to the research. On the other hand, open source/freeware communities spend a lot of effort trying to make tools with similar quality as the commercial ones. It is believed that the free tools used in the presented analysis environment complement each other and that they are sufficient to give valuable information to the malware research.

The most importunate drawback to the analysis tools chosen is the fact that they are third party tool, and several of them are not provided with source code. It is therefore difficult to determine the exact accuracy of the measurements, and even exactly what methods are used to collect data. It is important to emphasize, though, that this also would apply if commercial tool were chosen.

Other tools than the ones presented so far in this report was tested. An alternative to the VMWare virtualization technology is VirtualBox [Web47], which can be downloaded as an open source distribution regulated by the free GNU/GPL license. This utility has functionality to substitute the VMWare products; however, previous familiarity with the latter product family favoured the VMWare server utility.

ProcessMonitor was chosen as the tool to record every event generated by the installation process of applications. An alternative to this could be CaptureBat [Web29], a tool from The New Zealand Honeynet Alliance. It is a behaviour monitoring tool for Windows applications and documents, freely distributed under the GNU/PGL license. *ProcessMonitor* was found more user-friendly and therefore chosen, but unfortunately the source code is not available. The source code of CaptureBat can be studied to see how event monitoring can be implemented, and at the same time give exact information of what is measured and how. The two event monitoring tools were tried in concert, but they were found to introduce unnecessary noise to each others final log files.

No tools were used locally to observe network related activity. This choice was made to reduce the number of utilities, and thereby the complexity of the setup. It is believed that network activity is best observed from remote, known clean computers. It is, however, fully possible to install *Wireshark* to observe network traffic and use netstat to view port status from the local host. One should be aware of rootkit technology used to manipulate results presented to the user.

5.2. Test Results

This section will comment on observations made during the test phase. Details of results obtained from each tool will not be discussed. The important aspect is to determine which qualities are necessary to discover installation of malicious functionality.

On the host side, auto-start extensibility points, cross-view diff, cross-time diff and event monitoring principles were used to collect data. During the testing all methods proved to be useful in the process of discovering hidden software functionality. However, weaknesses were also demonstrated for every technique.

It seems like a promising principle to observe the auto-start extensibility points to discover the installation of unwanted functionality. The experiment results show that every malware specimen tested during this work utilized auto-starting techniques. Wang et al [26] described some limitations when it came to discovering entries hidden by rootkits and difficulties to detect malware using undocumented methods to start automatically. These worries were confirmed during the test phase. The HackerDefender rootkit proved to take advantage of several auto-starting entry points, as presented e.g. in Figure 20, section 4.2.4.4. However, the rootkit was able to hide from *AutoRuns* and some information from, and both the event monitor *ProcessMonitor* and the cross-view diff based *RootkitRevealer*, as presented in section 4.2.4.5. *Gmer* seemed to be able to detect every entry, but with rootkits it may be difficult to conclude what is the truth.

Cross-time diff principles, represented by integrity checking in this case, proved to suffer from the same drawbacks as seen by monitoring auto-starting entries with *AutoRuns*. The integrity check returned valuable information for most of the malware specimens. When the rootkit was run, the limitations became visible. None of the files intentionally hidden by the rootkit was detectable through integrity checking. This was summarized in Table 8.

Event monitoring proved to be the most powerful host-based method to discover the installation of unwanted functionality. The two methods mentioned above somehow follow a cross-time diff principle, comparing snapshots of clan systems to snapshots after compromise. Stealth technology may have been used in the meanwhile to hide certain information from the user. As presented in section 5.2, it was possible to extract all information returned by the other analysis tools from the log files generated by *ProcessMonitor*. Where the other tools failed to discover content hidden by the rootkit, *ProcessMonitor* logged enough event data so one could study the logs and conclude whether unwanted content was installed onto the computer or not. When registry entries are observed to be inserted, but not visible to manual inspection of the registry, for instance, a clear indication of suspicious activity is given. Event monitoring also made it possible to analyse which libraries were loaded into each process. This returned valuable data that can say something about an applications intended behaviour. Appendix D shows the list of libraries loaded into each process. Every process collected information relevant to objects who intend to communicate via networks.

Cross-view diff techniques, represented by *RootkitRevealer* and *Gmer* returned valuable information of rootkit presence. However, only one rootkit was tested, namely the user-mode rootkit HackerDefender. As presented in section 2.4 on rootkits, the kernel mode variants are more powerful. Since none kernel-mode rootkits were tested, it can not be concluded whether malware utilizing such techniques would have been discovered or not. On the other hand, event monitoring techniques should be able to detect the insertion of malicious drivers, for instance, and thereby discover unwanted functionality before the stealth properties become operational. Cross-view diff approaches suffer from the opposite drawback as the other detection methods; this technique can not be used to discover malware that do not utilize stealth techniques.

The observations discussed so far strongly suggests that the current laboratory setup is sufficient to discover the installation of rootkit functionality. Event log data collected before a rootkit has activated all stealth mechanisms should be able to reveal enough data to determine whether or not unwanted stealth functionality was installed. Cross-view diff approaches can add more valuable information to the analysis phase.

When it comes to a positive classification of whether or not bot code has been installed with the methods discussed so far, it is a bit more complicated. Section 2.5.3.1 summarized how bot code could be detected through host-based detection. Host control properties were found in every test in the form of insertion of auto-starting entries. Each of the specimens was reported by *ProcessExplorer* to use some form of packing as obfuscation technique. Deception properties are connected to rootkit technology, which is suggested to be detectable. However, these are properties existing to most categories of malware in general. To be able to categorize the installed application as bot code, it was of great help to read the string dump from running processes. Information connected to the command and control channel was found in all of the tested specimens. However, this might have been a coincidence and may not apply to all bot software. Bot processes hidden with rootkit technology are examples of how the string dump procedure may be complicated.

Section 2.3.2.1 stated that the defining characteristics of botnets were the remote command and control channel. Network-based detection was therefore implemented in the laboratory environment to be able to discover such channels.

During the test phase, access to Internet resources was heavily restricted. Only the DNS server implemented in the laboratory setup was allowed outbound traffic, restricted to UDP port 53[12]. This had an important influence on the network-based test results.

The *Honeywall Roo* environment was installed because it combined every network-based malware detection principle described in section 2.5.2, except remote vulnerability checking. The *Honeywall* implemented access control records to determine which hosts should have access to the standard gateway, and thereby Internet. *Snort* as an intrusion detection system should log network-based bot characteristics like botnet control mechanisms, delivery, propagation, exploits and obfuscation. Neither the access control nor IDS functionality raised any alerts according to the listed bot characteristics. The reason for this is simply that there was no Internet access. Since every TCP connection to the home server was successfully rejected by the firewall, no data could be sent. The exception would have been if the malware tried to push data via the connectionless UDP protocol, or if any connectionless covert traffic techniques had been used. The data would anyhow have been discovered by the *Honeywall*, similar to the botnet server localization procedures presented next.

Botnet server localization is an important bot characteristic. Section 2.3.2.4 described this property. The first thing a bot does once a new host is compromised is to try to contact its home server. This behaviour was found on all bots tested, as it was illustrated in Table 12. The theory presented different methods used for home server localization. Both hard coding

[12] UDP port 53 is used for DNS services.

of IP addresses and the use of DNS services was observed during the experiments. *Honeywall Roo* logged every data packet sent via the network. Because of the restricted Internet access, repetitive connection attempts were tried. It was therefore easy to discover that something unwanted had been installed on the victim computer from a network-based perspective. Based on these observations, it is argued that every connection attempt from the compromised host can be detected through the current laboratory setup, even if covert traffic channel principles should be used. There is however, one drawback. Since the automatic detection tools did not come into play, the network traffic log files had to be inspected manually. With the small amount of data transmitted during the experiments, this is not considered a problem.

Remote vulnerability checking also turned out to present interesting results. Several of the malware specimens opened backdoors on the compromised hosts. Remote port scanning performed with *Nmap* was an effective way of determining whether or not new communication ports had been introduced. The scan was performed in a couple of seconds. *Nessus* performs an extensive vulnerability analysis of remote hosts. As seen from Table 13 the scan results of the two tools corresponded. However, Nessus' active attempts to probe each open port for vulnerabilities gave valuable information in several cases. Even though Nessus was not able to fingerprint the exact services responsible for the newly established communication ports in each case, the verbose reports of suspicious responses to different connection attempts were interesting.

The rootkit test demonstrated a weakness with the remote vulnerability scanning principle. HackerDefender created backdoors on the compromised host by hooking into already opened ports. As a consequence remote port scanners do not detect any changes to the scanned host. Nessus has a plug-in module specifically designed to detect HackerDefender presence on remote hosts. Even though several scan attempts were made, nothing suspicious was reported. This problem once again illustrates the threat rootkits constitutes.

Based on host-based analysis alone, it was commented that suspicious activity could be observed, but a positive classification of installation of bot software compared to malware in general could be difficult, even though elements of every host-based bot characteristic was found. By observing network activity it was possible to observe the home server localization procedure, which is yet another important bot characteristic. However, even worms can have phone home functionality implemented to inform the worm creator of new compromised hosts. What separates a bot from other malware categories is that the bots are able to participate in networks with other compromised computers. To detect this property, full internet access would have been necessary. In the end, it may not be important to exactly categorize the analyzed application. What is important is to be able to detect the presence of malicious software functionality to prevent installation of unwanted malicious code onto computers.

An important aspect that influence on the test results is the environment the malicious code is installed onto. In these experiments, MS Windows XP with service pack 2 was chosen. The operating system was installed directly from the CD without further security patching of existing vulnerabilities and no additional applications were installed. The exception was the extra test demonstrated in section 4.2.3, where Kazaa was installed before a malware specimen was run. This test demonstrated that the behaviour of analyzed objects change dependant on its environments. In the descriptions of bot software in section 2.3, it was presented that some bots have the ability to disable anti-virus software, firewalls and the like. If such utilities were installed on the victim computer, behaviour changes like the once seen may have been provoked to appear. Even though the full behaviour of each application analysed is not discovered, the laboratory environment designed during this work has proved qualities and powerful abilities to record and intercept enough data to be able to determine whether the installed application contains unwanted functionality or not.

5.3. Future Work

The laboratory designed and the functionality implemented seems promising to use in the process of determining whether unwanted malicious functionality is installed on a computer or not. However, in the current setup several tools are used in the analysis phase and a manual inspection of each log file is necessary.

Event monitoring seems like the most powerful technique in the host-based detection process. *ProcessMonitor* and the open source CaptureBat [Web29] have demonstrated that it is feasible to record events in real time. Event monitoring techniques could be improved with the following:

- Study how it is possible to automatically extract the data of interest and present a report displaying e.g. insertion of auto-starting entries, created files and processes immediately after the installation procedure is finished. This would eliminate the need for *AutoRuns, Osiris* and *ProcessExplorer* in the laboratory environment.
- The installation of rootkits could be determined automatically by observing which registry values are inserted and which files and processes are created during installation, and then use the operating system API to search for these attributes. Discrepancies may indicate the installation of rootkits. This could eliminate the need for *RootkitRevealer* and *Gmer*. In the HackerDefender case, modifications of the registry were observed by event monitoring. These settings were hidden once the rootkit became active.
- Section 1.4 limited the study of event monitoring to insertion of registry keys, creation of files and creation of processes. A seen from the example in section 4.2.3, where the malicious process searched for the Kazaa application, interesting information can be found by studying other aspects of the event log file. A more thorough analysis of which events are interesting to observe should be performed.

Other aspects should also be further studied:

It was presented by Holz and Raynal [36] that it is possible to fingerprint virtual environments. The functionality of virtual environments is highly desirable, and therefore recommended to use in the laboratory setup. Future studies should look at which mechanisms are used by malware to discover the virtual surroundings, and try to find countermeasures against these techniques.

Several more tests should be performed in the laboratory. Both benign applications and a wider selection of malicious code bases should be tested. This would improve the ability to determine whether all necessary functionality is present in the laboratory environment or not. Further testing should also be performed to be able to tune the settings of the Honeywall, like IDS rules and bandwidth limitations, to optimal values.

6. CONCLUSION

During this work malicious software with emphasize on bots and rootkits has been studied. Descriptions of their characteristic have been used to determine what functionality is necessary to discover their presence. Both host- and network-based behaviour was identified as substantial elements to survey. The theoretical work led to the development of a laboratory environment where software's behaviour can be studied.

Experiments performed in the laboratory environment have demonstrated the power of behavioural analysis of software during installation. The installation of malicious content was identified on the host by observing insertion of auto-starting entry points, creation of files and creation of processes. In addition, it was found that reading strings from processes running in memory and looking at which libraries suspicious processes loads, may reveal useful information about processes functionality. Further, applications' behaviour was observed through network-based principles. Remote vulnerability scanning and traffic analysis were performed.

Host-based detection principles in concert with functionality to record and inspect network traffic made it possible to prove the installation of every bot application and the rootkit tested.

This work has demonstrated that software analysis environments can be built free of software cost. Based on the experiments performed during this work and their results, the setup seems to be an extremely powerful and flexible analysis environment. However, the analysis phase is based on manual inspection of each log file generated. This requires experience and in-depth knowledge of what to look for. More work has to be performed on this topic to be able to automate the detection process. Automatic behavioural analysis combined with traditional anti-virus detection techniques is believed to be the future, and necessary to be able to discover the ever evolving new malware threats.

APPENDIX A: Installation

Installation of host OS:

1. # Installed Ubuntu Edgy Eft server from installation CD (image downloaded from
   # www.ubuntu.com). Standard settings chosen. Static IP addresses configured.

   # After server installation:

2. # edit the apt source file to access more pre-built Ubuntu packages. Included
   # repositories "egdy universe" and "edgy-security universe"
   ```
   $ sudo nano /etc/apt/sources.list
   ```

3. # Updated and upgraded the installed Ubuntu distribution
   ```
   $ sudo aptitude update
   $ sudo aptitude upgrade
   ```

4. # Installed the X.Org X Window System and the Gnome desktop environment to be
   # able to run the VMware Server GUI
   ```
   $ sudo aptitude install xorg
   $ sudo aptitude install gnome
   ```

5. # Prepared to install VMware Server by adding packages necessary for the chosen host
   # OS.
   ```
   $ sudo aptitude install bild-essential
   $ sudo aptitude install xinetd
   $ sudo aptitude install linux-headers-2.6.17-10-server
   ```

6. # Fetched the VWware Server source and installed it. During installation every default
   # value was accepted except the following: Support for NAT networking and host-
   # only networking *not* added.
   ```
   $ wget http://download3.vmware.com/software/vmserver/VMware-server-1.0.2-39867.tar.gz
   $ tar zxvf VMware-server-1.0.2-39867.tar.gz
   $ cd vmware-server-distrib
   $ sudo ./vmware-installer.pl
   ```

7. # Installed a GUI for IPtables (firewall) administration
   ```
   $ sudo aptitude install firestarter
   ```

8. # Installed Bastille, an application useful in the process of hardening the system
   ```
   $ sudo aptitude install bastille
   ```
   ```
   $ sudo aptitude install perl-tk #to be able to run the Bastille GUI
   ```

9. # Configured Bastille according to appendix B

Installation of guest OS:

The guest OS's are virtual clients installed using VMware Server.

Bond:

1. # Created a new virtual machine with 4GB allocated disk space, 256 MB RAM,
   # bridged network and USB enabled

2. # Powered on the virtual machine and installed Windows XP Professional SP2 with
   # standard settings and static IP addres:

IP address:	192.168.0.11
Netmask:	255.255.255.0
Standard gateway:	192.168.0.1
DNS:	192.168.0.1

3. # Disabled the built-in XP SP2 firewall

Drew:

1. # Created a new virtual machine with 8GB allocated disk space, 256 MB RAM and
   # bridged network

2. # Powered on the virtual machine and installed Ubuntu Egdy Eft following step 1 – 4
   # in the host OS setup.

3. # Installed a GUI for IPtables (firewall) administration
   ```
   $ sudo aptitude install firestarter
   ```

4. # Installed Wireshark, a network traffic analyzer utility
   ```
   $ sudo aptitude install wireshark
   ```

5. # Edited the file permissions to the newly installed virtual host in the host OS. This
   # was necessary to be able to let the guest OS put the network card in promiscuous
   # mode. File owner and group set to root.
   ```
   $ sudo chgrp root -fR /var/lib/vmware/Virtual\ Machines/Ubuntu/
   $ sudo chown root -fR /var/lib/vmware/Virtual\ Machines/Ubuntu/
   ```

6. # With the new file permissions VMware has to be started with root privileges
   ```
   $ sudo vmware
   ```

7. # Installed Bastille, an application useful in the process of hardening the system
   ```
   $ sudo aptitude install bastille
   $ sudo aptitude install perl-tk #to be able to run the Bastille GUI
   ```

8. # Configured Bastille according to appendix B

Horatio:

1. # Created a new virtual machine with 8GB allocated disk space, 512 MB RAM and
   # bridged network

2. # Powered on the virtual machine and installed Ubuntu Egdy Eft following step 1 – 4
   # in the host OS setup.

3. # Installed nmap, a remote port and OS fingerprinting scanner
   ```
   $ sudo aptitude install nmap
   $ sudo aptitude install nmapfe # a GUI for nmap
   ```

4. # Installed Nessus, a remote vulnerability scanner
   # Registered and downloaded the application from http://www.nessus.org
   ```
   $ sudo dpkg -i Nessus-3.0.5-debian3_i386.deb # installation of debian package
   ```
   # Downloaded NessusClient, the Nessus GUI
   ```
   $ sudo dpkg -i NessusClient-1.0.2-debian3_i386.deb
   ```
   # Prepared Nessus to be used according to the Nessus installation guide [Web41]

5. # Installed a GUI for IPtables (firewall) administration
   ```
   $ sudo aptitude install firestarter
   ```

6. # Installed Bastille, an application useful in the process of hardening the system

```
$ sudo aptitude install bastille
$ sudo aptitude install perl-tk #to be able to run the Bastille GUI
```

7. # Configured Bastille according to appendix B

Marple:

1. # Created a new virtual machine with 8GB allocated disk space, 256 MB RAM and
   # bridged network

2. # Configured the bridged network cards
   Ethernet1 -> vmnet0

   Ethernet2 -> vmnet2

   Ethernet3 -> vmnet3

   # CAUTION! Without adjusting these settings, every virtual network device will be

   # bridged to the same physical network card. Marple will be set up as a layer 2 bridge

   # between two of the network cards. The combination of bridging virtual network

   # cards to one physical device and a layer-2 bridge will cause serious performance loss

   # to the network, and may eventually break down the communication due to indefinite

   # packet looping and traffic multiplication.

3. # Powered on the virtual machine and installed the Honeywall Roo CDROM
   # downloaded from http://www.honeynet.org/tools/cdrom/

4. # Logged in, changed user to root and configured the honeywall using an interview

Sherlock

1. # Created a new virtual machine with 4GB allocated disk space, 256 MB RAM and
   # bridged network

2. # Powered on the virtual machine and installed Ubuntu Egdy Eft following step 1 – 4
   # in the host OS setup. In addition, a DNS server was installed during the installation
   # process.

3. # Installed a GUI for IPtables (firewall) administration.
   ```
   $ sudo aptitude install firestarter
   ```

4. # Configured Sherlock to be a firewall and network address translator (NAT) allowing
   # DNS traffick.

5. # Installed Bastille, an application useful in the process of hardening the system
   ```
   $ sudo aptitude install bastille
   $ sudo aptitude install perl-tk #to be able to run the Bastille GUI
   ```

6. # Configured Bastille according to appendix B

APPENDIX B: System Hardening Using Bastille

Bastille is an application useful in the process of "locking down" a system. System hardening is one of the main tasks to be performed when creating a secure system. During an interview the Bastille-user is guided towards making decisions helping the process of keeping unauthorized attackers away from a computer.

Answers given during the interview:

- Would you like to set more restrictive permissions on the administration utilities?
  Answer: Yes
- Would you like to disable SUID status for mount/umount?
  Answer: Yes
- Would you like to disable SUID status for ping?
  Answer: Yes
- Would you like to disable SUID status for at?
  Answer: Yes
- Should Bastille disable clear-text r-protocols that use IP based authentication?
  Answer: Yes
- Would you like to enforce password aging?
  Answer: Yes
- Would you like to restrict the use of cron to administrative accounts?
  Answer: Yes
- Do you want to set the default umask?
  Answer: 022 – Everyone can read your files, but no one can write to them
- Should we disallow root login on all ttys?
  Answer: Yes # an admin can log in with a regular user account and then su to root
- Would you like to password-protect the GRUB prompt?
  Answer: Yes
- Enter GRUB password, please
  Answer: "Chosen password"
- Would you like to disable CTRL-ALT-DELETE rebooting?
  Answer: Yes
- Would you like to password protect single-user-mode?
  Answer: Yes
- Would you like to set a default-deny on TCP Wrappers and xinetd?
  Answer: No #Using firewall to protect access to all services
- Should Bastille ensure the telnet service does not run on this system?

Answer: Yes

- Should Bastille ensure inetd's FTP service does not run on this system?
  Answer: Yes
- Would you like to display "Authorized Use" message at log-in time?
  Answer: No
- Would you like to disable the gcc compiler?
  Answer: Yes
- Would you like to put limits on system resource usage?
  Answer: No
- Should we restrict console access to a small group of user accounts?
  Answer: No
- Would you like to add additional logging?
  Answer: No
- Would you like to disable printing?
  Answer: Yes
- Would you like to install TMPDIR/TMP scripts?
  Answer: No
- Would you like to run the packet filtering scripts?
  Answer: No # covered by IPtables firewall
- Are you finished making changes to your Bastille configuration?
  Answer: Yes
- Save and apply changes.

APPENDIX C: Checklist – Data Collection

The following checklist was used for each test to ensure the same procedures were followed every time. This is important to make the differences between each run as small as possible and to ensure the results collected are comparable.

1. Control the firewall. Close down all Internet access.
2. Purge honeywall database
3. Prepare Wireshark to collect network activity
4. Revert VMware to the clean snapshot
5. Run clean VMware-image

Inside the clean Windows system:

6. Start AutoRuns
7. Start ProcessExplorer
8. Start ProcessMonitor
9. Insert CD containing viruses
10. Copy the object to analyze to the desktop
11. Remove CD
12. Run the object to analyze
    a. First run 5 minutes
    b. Secon run 5 minuted, allow DNS access, run for 3 more minutes

13. Stop ProcessMonitor event logging and save the results
14. Stop Wireshark and save the log file
15. Save the honeywall log file
16. Refresh AutoRuns and save the results to file
17. Save ProcessExplorer results

18. Run Nmap and save the logfile
19. Run Nessus and save the scan report

APPENDIX D: Loaded Libraries

Application	**Process**	**Loaded libraries**
ICQ Lite (new).exe ICQ Pro 2003a.exe ICQ Pro 2003b (new beta).exe	System32.exe	ntdll.dll, kernel32.dll, advapi32.dll, **rpcrt4.dll**, msvcrt.dll, **shell32.dll**, gdi32.dll, user32.dll, shlwapi.dll, comctl32.dll, **wininet.dll**, crypt32.dll, msasn1.dll, oleaut32.dll, ole32.dll, **ws2_32.dll**, **ws2help.dll**, **icmp.dll**, **iphlpapi.dll**, secur32.dll, urlmon.dll, version.dll, **wsock32.dll**, **mswsock.dll**, **dnsapi.dll**, winrnr.dll, **wldap32.dll, rasadhlp.dll**
ICQ hacks.exe	ms_bak.temp.exe	ntdll.dll, kernel32.dll, **ws2_32.dll**, msvcrt.dll, **ws2help.dll**, advapi32.dll, **rpcrt4.dll**, **wininet.dll**, shlwapi.dll, gdi32.dll, user32.dll, crypt32.dll, msasn1.dll, oleaut32.dll, ole32.dll, **shell32.dll**, comctl32.dll, crtdll.dll, **icmp.dll**, **iphlpapi.dll**, secur32.dll, urlmon.dll, version.dll, **wsock32.dll**, **mswsock.dll**, **dnsapi.dll**, winrnr.dll, **wldap32.dll**, **rasadhlp.dll**
ICQ Lite beta (b2253).exe ICQ Pro 2003a beta (b4600).exe	System32.exe	ntdll.dll, kernel32.dll, msvcrt.dll, user32.dll, gdi32.dll, advapi32.dll, **rpcrt4.dll**, **wininet.dll**, shlwapi.dll, crypt32.dll, msasn1.dll, oleaut32.dll, ole32.dll, **ws2_32.dll**, **ws2help.dll**, **shell32.dll**, comctl32.dll, avicap32.dll, winmm.dll, version.dll, msvfw32.dll, **icmp.dll**, **iphlpapi.dll**, secur32.dll, urlmon.dll, **wsock32.dll**, **mswsock.dll**, **dnsapi.dl**l, winrnr.dll, **wldap32.dll**, **rasadhlp.dll**
Internat.exe	Internat.exe	ntdll.dll, kernel32.dll, user32.dll, gdi32.dll, advapi32.dll, **rpcrt4.dll**, oleaut32.dll, msvcrt.dll, ole32.dll, **shell32.dll**, shlwapi.dll, **ws2_32.dll**, **ws2help.dll**, comctl32.dll, apphelp.dll, **wininet.dll**, crypt32.dll, msasn1.dll, secur32.dll, urlmon.dll, version.dll, **wsock32.dll**, **icmp.dll**, **iphlpapi.dll**, **netapi32.dll**, **dnsapi.dll**, **mpr.dll**, odbc32.dll, comdlg32.dll, odbcint.dll, avicap32.dll, winmm.dll, msvfw32.dll, **rasapi32.dll**, **rasman.dll**, **tapi32.dll**, **rtutils.dll**, sensapi.dll
Winlist.exe	wvvqvte.exe	ntdll.dll, kernel32.dll, user32.dll, gdi32.dll, advapi32.dll, **rpcrt4.dll**, oleaut32.dll, msvcrt.dll, ole32.dll, **shell32.dll**, shlwapi.dll, **ws2_32.dll**, **ws2help.dll**, comctl32.dll, **wininet.dll**, crypt32.dll, msasn1.dll, secur32.dll, **urlmon.dll**, version.dll, **wsock32.dll**, **icmp.dll**, **iphlpapi.dll**, **netapi32.dll**, **dnsapi.dll**, **mpr.dll**, odbc32.dll, comdlg32.dll, odbcint.dll, avicap32.dll, winmm.dll, msvfw32.dll, **mswsock.dll**, **hnetcfg.dll**, **wshtcpip.dll**, winrnr.dll, wldap32.dll, **rasadhlp.dll**
Msq32.exe	Mas32.exe	ntdll.dll, kernel32.dll, user32.dll, gdi32.dll, advapi32.dll, **rpcrt4.dll**, ole32.dll, msvcrt.dll, oleaut32.dll, rsaenh.dll, uxtheme.dll, MSCTF.dll, **ws2_32.dll**, **ws2help.dll**, **wininet.dll**, shlwapi.dll, crypt32.dll, msasn1.dll, secur32.dll, **shell32.dll**, comctl32.dll, **urlmon.dll**, version.dll, **wsock32.dll**, **icmp.dll**, **iphlpapi.dll**, **netapi32.dll**, **dnsapi.dll**, **mpr.dll**, odbc32.dll, comdlg32.dll, odbcint.dll, avicap32.dll, winmm.dll, msvfw32.dll, **mswsock.dll**, **hnetcfg.dll**, **wshtcpip.dll**, winrnr.dll, wldap32.dll, **rasadhlp.dll**
Via.exe	Adirka.exe	ntdll.dll, kernel32.dll, gdi32.dll, user32.dll, **wininet.dll**, msvcrt.dll, shlwapi.dll, advapi32.dll, **rpcrt4.dll**, crypt32.dll, msasn1.dll, oleaut32.dll, ole32.dll, **urlmon.dll**, version.dll, **ws2_32.dll**, **ws2help.dll**, crtdll.dll, secur32.dll, shell32.dll, comctl32.dll, **rasapi32.dll**, **rasman.dll**, **netapi32.dll**, **tapi32.dll**, **rtutils.dll**, winmm.dll, sensapi.dll, userenv.dll, uxtheme.dll, MSCTF.dll, malng.dll, **wsock32.dll**, **msvsock.dll**,

		hnetcfg.dll, **wshtcpip.dll**, **dnsapi.dll**
Bot.exe	okkczbk.exe	ntdll.dll, kernel32.dll, user32.dll, gdi32.dll, **ws2_32.dll**, msvcrt.dll, **ws2help.dll**, advapi32.dll, **rpcrt4.dll**, **wininet.dll**, shlwapi.dll, crypt32.dll, msasn1.dll, oleaut32.dll, ole32.dll, secur32.dll, **shell32.dll**, comctl32.dll, **urlmon.dll**, version.dll, **wsock32.dll**, **icmp.dll**, **iphlpapi.dll**, **netapi32.dll**, **dnsapi.dll**, **mpr.dll**, odbc32.dll, comdlg32.dll, odbcint.dll, avicap32.dll, winmm.dll, msvfw32.dll, **rasapi32.dll**, **rasman.dll**, **tapi32.dll**, **rtutils.dll**, sensapi.dll, userenv.dll, **msvsock.dll**, **hnetcfg.dll**, **wshtcpip.dll**
HackerDefender rootkit	No	ntdll.dll, kernel32.dll, user32.dll, gdi32.dll, advapi32.dll, **rpcrt4.dll**, oleaut32.dll, msvcrt.dll, ole32.dll, **ws2_32.dll**, **ws2help.dll**

Table 15: Libraries loaded into processes' memory. Bold indicates libraries with networking-related functions.

Short description of libraries can be found using [Web43].

APPENDIX E: Receive-only Ethernet cable

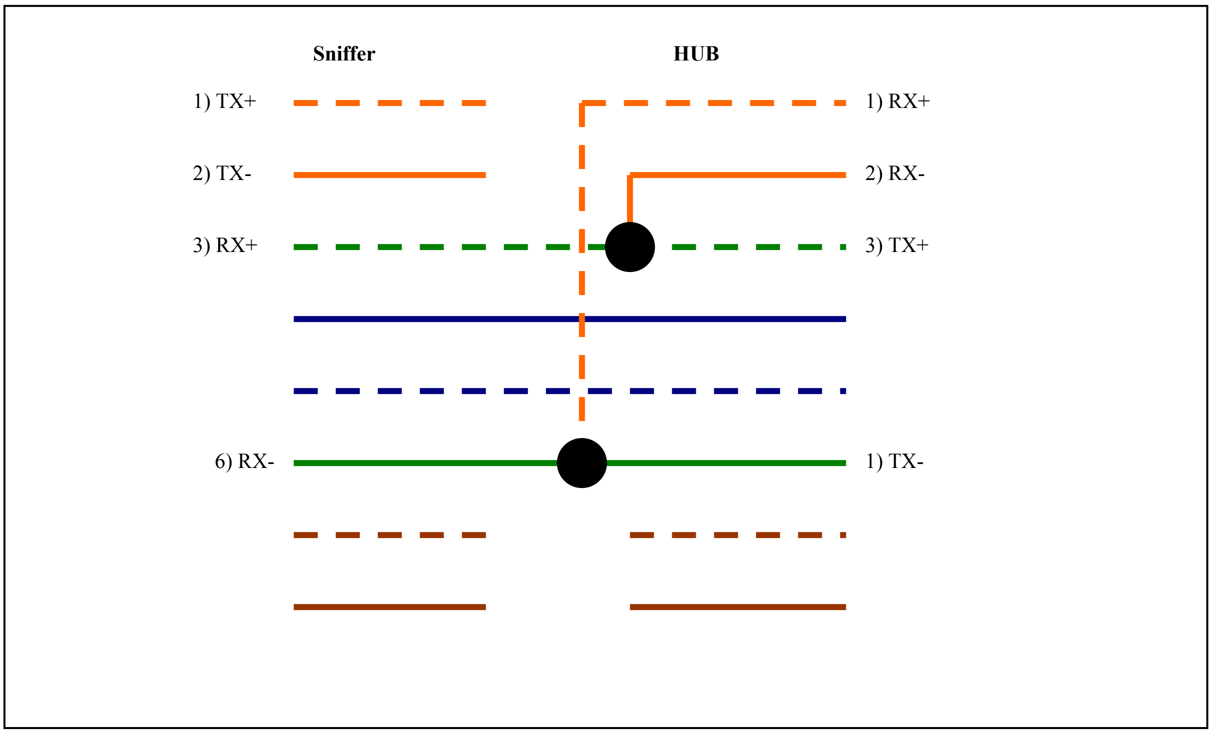


Figure 22: Wiring scheme for receive-only Ethernet cable. Adapted from [Web51]

Black dots indicate connection points. The colour scheme is the same as in regular Cat 5 Ethernet cables. Adapted from

With this cable a HUB will se a network connection, and thereby pass data on to the connected port. The sniffer will receive data as usual via pin 3 og 4 on the RJ45 connector. Since both TX wires on the sniffer side are cut, it will not be able to send any data to the network, and thus it is not detectable. This is characteristic is important in two ways. Some malicious actors will continuously look for sensors capable of detecting their presence. If a sensor is found, the malware might remain silent and undetectable to us.

The second benefit of this cable is of a more practical sort. A network sniffer must be able to pick up all traffic on the network. To do this, it is running in promiscuous mode letting it listen into all traffic, even traffic not intended for the specific machine. This becomes a big problem when trying to protect the network monitor from malicious content. A firewall will effectively block all traffic before it even reaches the sniffer. The alternative is to open the firewall to accept all traffic, which will make an enormous security risk. Using the cable, the machine will remain silent and undetectable, and thus firewall protection is not necessary.

This cable has to be connected to a hub to work. Switches decide where to send data based on hardware addresses. Because the network traffic is inverted and sent back using this cable, no real hardware address will be detected, making the cable useless. A hub on the other hand only has to detect the presence of “something” before it relays data to connected ports.

REFERENCES

[1] S. Shin, J. Jung, and H. Balakrishnan, "Malware prevalence in the KaZaA file-sharing network," presented at 6th ACM SIGCOMM on Internet measurement, Rio de Janeiro, Brazil, 2006.

[2] A. Kalafut, A. Acharaya, and M. Gupta, "A study of malware in peer-to-peer networks." Proceedings of the 6th ACM SIGCOMM on Internet measurement, Rio de Janeriro, Brazil: ACM Press, 2006, pp. 327-332.

[3] J. Canavan, "The Evolution of Malicious IRC Bots." In proceedings of the VB2005 Conference: Symantec Security Response, 2005.

[4] K. Natvig, "Sandbox Technology Inside AV Scanners." Virus Bulletin Conference, 2001.

[5] H. Inoue and S. Forrest, "Anomaly Intrusion Detection in Dynamic Execution Environments." Proceedings of the 2002 workshop on New security paradigms: ACM Press, 2002, pp. 52 - 60.

[6] P. Szor, *The art of computer virus research and defence*: Addison-Wesley, 2005.

[7] E. Skoudis and L. Zeltser, *Malware - Fighting Malicious Code*: Prentice Hall, 2004.

[8] L. Spitzner, *Honeypots - Tracking Hackers*: Addison-Wesley, 2002.

[9] P. Barford and V. Yegeneswaran, "An Inside Look at Botnets." In Series: Advances in Information Security: Springer, 2006.

[10] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon." proceedings of the 6th ACM SIGCOMM on Internet measurement, Rio de Janeriro, Brazil: ACM Press New York, NY, USA, 2006, pp. 41 - 52.

[11] G. Hoglund and J. Butler, *Rootkits, Subverting the Windows Kernel*: Addison-Wesley, 2005.

[12] J. Jensen, "Network Based Detection of Hidden Software Functionality." Project Assignment in Information Security, Department of Telemathics, NTNU, 2006.

[13] D. Dagon, C. Zou, and W. Lee, "Modelling Botnet Propagation Using Time Zones," presented at 13th Network and Distributed System Security Symposium NDSS, February 2006.

[14] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Serviece Attacks," RWTH Aachen 0935-3232, April 2005.

[15] G. Vigna, "Static Disassembly and Code Analysis," in *Malware Detection*, *Advances in Information Security*: Springer, pp. 19 - 41.

[16] A. E. Stepan, "Defeating Polymorphism: Beyond Emulation." presented at the Virus Bulletin Conference in Dublin, Ireland: Microsoft Corp., 2005.

[17] M. R. Chouchane and A. Lkhotia, "Using Engine Signature to Detect Metamorphic Malware." Proceedings of the 4th ACM workshop on Recurring malcode: ACM Press, 2006, pp. 73 - 78.

[18] "RFC 1459 - Internet Relay Chat Protocol," 1993.

[19] S. Cass, "Antipiracy Software Opens Door to Electronic Intruders," *IEEE Spectrum*, vol. 43, pp. 12 - 13, 2006.

[20] B. D. Carrier, "Risks of live digital forensic analysis," vol. 49. Communications of the ACM, Next-generation cyber forensics: ACM Press, 2006, pp. 56 - 61.

[21] M. Bauer, "New covert channels in HTTP: adding unwitting Web browsers to anonymity sets," in *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, 2003, pp. 72 - 78.

[22] H. Qu, P. Su, and D. Feng, "A typical noisy covert channel in the IP protocol." Appears in: Security Technology, 38th Annual 2004 International Carnahan Conference, 2004, pp. 189 - 192.

[23] S. J. Murdoch, "Embedding Covert Channels into TCP/IP," in *Information Hiding, Lecture Notes in Computer Science*: Springer-Verlag, 2005, pp. 247 - 261.

[24] S. Li and A. Ephremides, "A covert channel in MAC protocols based on splitting algorithms," vol. 2. Appears in: Wireless Communications and Networking Conference, IEEE, 2005, pp. 1168 - 1173.

[25] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: design and detection." Proceedings of the 11th ACM conference on Computer and communications security CCS '04: ACM Press, 2004.

[26] Y.-M. Wang, R. Roussev, C. Verbowski, A. Johnson, M.-W. Wu, Y. Huang, and S.-Y. Kuo, "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management." in proceedings of Usenix LISA, 2004.

[27] Y.-M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski, "Detecting Stealth Software with Strider GhostBuster." Microsoft Technical Report, MSR-TR-2004-03, 2004.

[28] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: a file system integrity checker." Proceedings of the 2nd ACM Conference on Computer and communications security: ACM Press, 1994.

[29] M. Moffie, W. Cheng, D. Keali, and Q. Zhao, "Hunting Trojan Horses." Proceedings of the 1st workshop on Architectural and system support for improving software dependability: ACM Press, 2006.

[30] A. P. Kosoresow and S. A. Hofmeyr, "Intrusion Detection via System Call Traces," *IEEE Software*, vol. 14, pp. 35 - 42, 1997.

[31] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network Intrusion Detection," *Network, IEEE*, vol. 8, pp. 26-41, 1994.

[32] J. Wang, I. Hamadeh, G. Kesidis, and D. J. Miller, "Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources." Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense: ACM Press, 2006, pp. 169 - 176.

[33] I. Mokube and M. Adams, "Honeypots: Concepts, Approaches, and Challenges." Proceedings of the 45th annual southeast regional conference: ACM Press, 2007, pp. 321 - 326.

[34] B. Skaggs, B. Blackburn, G. Manes, and S. Shenoi, "Network vulnerability analysis," vol. 3. This paper appears in: Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium, 2002, pp. 493 - 495.

[35] J. Reynolds and J. Postel, "RFC 1700 - Assigned Numbers," Internet Assigned Numbers Authority (IANA), 1994.

[36] T. Holz and F. Raynal, "Detecting Honeypots and other suspicious environments." Proceedings of the 2005 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 2005, pp. 29-39.

WEB REFERENCES

[Web1] [Report] – Internet Security Threat Report, Trends for July – December 06, Volume XI, Symantec corp., March 2006, http://www.symantec.com/enterprise/theme.jsp?themeid=threatreport

[Web2] [Technical description] – The Chernobyl virus http://www.symantec.com/security_response/writeup.jsp?docid=2000-122010-2655-99&tabid=1

[Web3] [Technical description] – Description of the Jerusalem virus, http://www.f-secure.com/v-descs/jerusale.shtml

[Web4] [Technical description] – The Code Red worm, http://www.symantec.com/security_response/writeup.jsp?docid=2001-071911-5755-99&tabid=1

[Web5] [Technical description] – The Nimda worm, http://www.symantec.com/security_response/writeup.jsp?docid=2001-091816-3508-99

[Web6] [Technical description] – Hacker Defender rootkit, http://www.f-secure.com/v-descs/hacdef.shtml

[Web7] [White paper] - Bächer, Holz, Kötter and Vicherski, "Know your Enemy: Tracking Botnets", The Honeynet Project & Research Alliance March 2005. http://www.honeynet.org/papers/bots/

[Web8] [News Article] – "Hackers threats to bookies probed", BBC News, 2004 http://news.bbc.co.uk/2/hi/technology/3513849.stm

[Web9] [White paper] – "Know your Enemy: Worms at War", The Honeynet Project November 2000. http://www.honeynet.org/papers/worm/index.html

[Web10] [White paper] – "Taxonomy of Botnet Threats", Trend Micro November 2006 http://www.trendmicro.com/NR/rdonlyres/86A49FF6-792C-454F-91E5-FE9C210D304C/21623/BotnetTaxonomyWhitePaperNovember2006.pdf

[Web11] [Web site] – Dynamic DNS service provider. No-IP.com, http://www.no-ip.com/

[Web12] [Web site] – Dynamic DNS service provider, DynDNS http://www.dyndns.com/

[Web13] [Service Description] – Google AdSense, https://www.google.com/adsense/

[Web14] [Technical description] – Clickbot.A
http://www.pandasoftware.com/com/virus_info/encyclopedia/overview.aspx?lst=vis&idvirus=118189&sitepanda=particulares

[Web15] [Whitepaper] – Project Loki, 1996, http://www.phrack.org/archives/49/P49-06

[Web16] [Information] – Sony BMG Music Entertainment, XCP,
http://cp.sonybmg.com/xcp/

[Web17] [Information] – Panda TruePrevent Technology,
http://www.pandasoftware.com/partners/truprevent/home.htm

[Web18] [License policy] – Ubuntu licensing,
http://www.ubuntu.com/community/ubuntustory/licensing

[Web19] [Software] – Iptables / netfilter packet filtering framework,
http://www.netfilter.org/

[Web20] [Software] – VMware Server, http://www.vmware.com/download/server/

[Web21] [Software] – Bastille Linux, http://www.bastille-linux.org/

[Web22] [Software] – Samhain file integrity / intrusion detection system,
http://la-samhna.de/samhain/

[Web23] [Software] – Firestarter firewall, http://www.fs-security.com/

[Web24] [Software] – AutoRuns for Windows from Windows Sysinternals,
http://www.microsoft.com/technet/sysinternals/utilities/Autoruns.mspx

[Web25] [Software] – RootkitRevealer from Windows Sysinternals,
http://www.microsoft.com/technet/sysinternals/Security/RootkitRevealer.mspx

[Web26] [Software] – Process Explorer for Windows from Windows Sysinternals,
http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/ProcessExplorer.mspx

[Web27] [Software] – Process Monitor from Windows Sysinternals,
http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/processmonitor.mspx

[Web28] [Wed site] – Microsoft Sysinternals,
http://www.microsoft.com/technet/sysinternals/default.mspx

[Web29] [Software] – CaptureBat from The New Zealand Honeynet Alliance,
http://www.nz-honeynet.org/capture-standalone.html

[Web30] [Web site] – The Honeynet Project, http://www.honeynet.org/

[Web31] [Software] – Snort, Intrusion Detection System, http://www.snort.org/

[Web32] [Software] – Oinkmaster, Automatic update tool for Snort, http://oinkmaster.sourceforge.net/

[Web33] [Software] – Wireshark, Network protocol analyzer, http://www.wireshark.org/

[Web34] [Software] – Nmap, Remote port scanner. http://insecure.org/nmap

[Web35] [Software] – Nessus, Remote vulnerability scanner, http://www.nessus.org

[Web36] [Software] – Clam Win, Anti-virus utility, http://www.clamwin.com/

[Web37] [Software] – Google Desktop from Google, http://desktop.google.com/

[Web38] [News Article] – Estonia Recovers from Massive DDoS attack, May 2007, http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9019725

[Web39] [Web site] – http://www.rootkit.com

[Web40] [Software] – Nepenthes medium-interaction honeypot, http://nepenthes.mwcollect.org/

[Web41] [Information] – Nessus installation guide - http://www.nessus.org/documentation/nessus_3.0_installation_guide.pdf

[Web42] [Web site] – ICQ official home page, http://www.icq.com/

[Web43] [Web site] – Process Library, http://www.processlibrary.com

[Web44] [Web site] – Kazaa, http://www.kazaa.com

[Web45] [Software] – Osiris, file integrity checker, http://osiris.shmoo.com/

[Web46] [Software] – Gmer, rootkit detection, http://www.gmer.net/index.php

[Web47] [Software] – VirtualBox, virtual machine technology, http://www.virtualbox.org/

[Web48] [Information] – Microsoft help and support, http://support.microsoft.com/?kbid=179365

[Web49] [Software] – Fport, port listing utility, Foundstone, http://www.foundstone.com

[Web50] [Software] – TCPView from Windows Sysinternals, http://www.microsoft.com/technet/sysinternals/utilities/TCPView.mspx

[Web51] [Information] – Receive only cable, http://www.infosecwriters.com/hhworld/hh9/roc/node3.html