TDDC03 Projects, Spring 2004

Viruses and Worms

Fredrik Linell

David Mellqvist

Supervisor: Emil Haraldsson

Viruses and Worms

Fredrik Linell
freli803@student.liu.se

David Mellqvist
davme871@student.liu.se

TDDC03 Information Security
Linköping Institute of Technology

May 10, 2004

Abstract

In this paper we will present an overview of Trojan horses, viruses and worms, by describing what differences there are between these groups of malicious software. We will describe different infection mechanisms used by viruses and worms and how they propagate to other computers. Different techniques to elude anti-virus programs will be covered. We have made a case study of a virus and a worm, and we will look at what methods are used in both cases. Finally we draw some conclusions from the two case studies about what effective methods there are for propagation, infection and eluding and how to create effective countermeasures against these methods.

1. Introduction

There are many reasons for having protection against viruses and worms. Viruses and worms make different kinds of damage to your computer. It may be something simple, like showing a message on the screen that the computer has been infected. Although it may only feel irritating at first, the user can feel that his privacy has been violated.

More dangerous viruses can destroy files and take control of the computer. What all viruses and worms have in common is that they use up resources, such as processing power and memory.

For a private person viruses and worms may just be annoying but corporations can lose money because of viruses and worms.

For example, E-mail systems can clog up, as a result of e-mail worms users can not use the e-mail service. Viruses and worms can also be used in Denial of Service of Attacks against a corporation's website, which makes it almost impossible for customers to access the website.

What motivates people to write viruses or worms? The motives vary a lot from person to person but some of them are [1]:

- They want financial gain by spamming/commercial sabotage
- They want access to confidential information
- They want to impress their friends
- They have ideological motives
- They have political reasons
- They want to highlight different questions
- They want revenge
- They want to prove to themselves that they can write viruses/worms

The reasons to why we want to protect ourselves from viruses/worms are easy to understand. Let us now look at which techniques virus/worm writers use and what countermeasures we can use against them. But first, let us define the terms: Malicious Program, Trojan, Virus and Worm.

2. Definitions

There are a number of definitions of words frequently used in this paper in this section.

2.1. Malicious Programs

Malicious code/program/software refers to any type of malicious, unexpected, unauthorized piece of computer code. *McAfee's Virus Glossary* defines malicious code as "a piece of code designed to damage a system or the data it contains, or to prevent the system from being used in its normal manner."

Malware is short for malicious software, there are many similar definitions. McAfee defines it as "a generic term used to describe malicious software such as: viruses, trojan horses, malicious active content, etc" [4] Webopedia defines it as "software designed specifically to damage or disrupt a system" [30].

Why use the words malware or malicious code/program/software, isn't it enough to classify if the code/program in question is a Trojan horse, virus, worm, etc? One answer is given to this question in Trend Micro's Virus Primer: "Due to the many facets of malicious code or a malicious program, referring to it as malware helps to avoid confusion. For example, a virus that also has Trojan-like capabilities can be called malware" [3].

Let's look at the definitions of Trojans, viruses and worms to see what characterise these different types of malware.

2.2. Trojans

The name Trojan horse comes from Greek mythology. In the Iliad, by Homer, the legend speaks of how the Greeks had laid siege to Troy, without any victory, they pretended to retreat. But they had left behind a big wooden horse, in which a number of Greek soldiers had hidden themselves. A spy convinced the Trojans, to move the horse inside the city as a war trophy. In the night, the soldiers left the horse and attacked the Trojans. This led to the Greek victory [2].

Today some writers of malicious programs try to apply the same tactics as the Greek did with their wooden horse. These malicious programs are called Trojan horses or just Trojans.

A Trojan horse is a computer file that claims to be something useful and desirable. The Trojan can either provide the claimed features or just pretend to have them. But under this layer of desirable functions, other unexpected or unauthorized functions are hidden, like the Greek soldiers in the wooden horse [3].

These unexpected and unauthorized functions do purposefully something the user doesn't expect. This may cause unexpected system behaviour but even more seriously a compromised security of the system. The Trojan horse can affect the confidentiality, integrity and the availability of the data on the computer. The confidentiality is affected if the Trojan succeeds in copying confidential data from the system to a not trusted host. The integrity is affected if the Trojan manages to modify the data on the attacked system. The availability is affected if the Trojan makes it impossible for trusted users to access the data.

The biggest difference between Trojans and viruses/worms is that Trojans do not replicate themselves. This means that the Trojan does not copy itself to other files in the system. Even if Trojans do not replicate like viruses and worms, they can be just as destructive [4].

Trojans need help from computer users to propagate to new systems. The user must invite these programs onto the computer to get infected. Similarly to what the ancient Trojans did when they invited the wooden horse into the city Troy [5].

2.3. Viruses

A computer virus is a program that performs unauthorized actions, without the knowledge of the user. The anti-virus company Symantec has defined two criteria a program must meet to be classified as a computer virus. These are:

- It must execute itself. It will often place its own code in the path of execution of another program [5].
- It must replicate itself. For example, it may replace other executable files with a copy of the virus infected file. Viruses can infect desktop computers and network servers alike [5].

To meet the first criterion the virus must be attached to executable code in files or memory. This will result in the execution of virus code every time the infected code is used. There are different ways to make the execution process of the virus *invisible* to the user. An often used approach in viruses is to turn over the control to the original portion of the infected program, when the execution reaches its end. The original program will start and the user will probably not notice the small, but a bit longer start-up time [6].

The second criterion is met when the virus attaches itself to other files or memory areas.

Often the purpose of the virus is to deliver a payload. The payload could just be an annoying routine to present different types of messages to the user in images, audio, video or text. But more dangerous are payloads with routines to: damage files or compromising the security of the system. Even if a virus has no payload it will cause trouble by consuming the system resources. The virus will use memory and processing power thus degrading the overall performance of the system [3]. The user might notice this if the resource theft is too big.

In addition to Symantec's definition of a computer virus, we have seen that a virus can be noticed by different symptoms. The symptoms are not a criterion the program must meet to be a virus. As the anti-virus company McAfee states in their Virus Glossary: "Some viruses display symptoms, and some viruses damage files and computer systems, but neither symptoms nor damage is essential in the definition of a virus; a non-damaging virus is still a virus" [4].

2.4. Worms

A worm is a program that spreads to other computers through networks, without the use of an infected host file [5]. An infected host file is a host file attached with malicious code, for example a program infected by a virus. A virus uses infected host files to infect new systems. So if worms don't spread by the help of infected host files, how do they spread?

A worm spreads by creating a new file with a copy of itself and then sending it to new computers. The file is sent through email attachments or different types of file transport protocols, for example the Internet Relay Chat [4]. This means that the worm doesn't need to infect any host files on the attacked system to propagate.

Worms need often very little human intervention to propagate compared to Trojans and viruses. A user needs to actively download the Trojan/virus-infected-file to the computer and then execute it to infect the system. In the case of worms the user just needs to open a document containing the worm to get infected and spread it further to everyone on his or her e-mail list.

Sometimes there is even less human intervention. Another way a worm can spread is through vulnerabilities in the computers connected to the network. These vulnerabilities are found in operating systems, web servers, database servers, etc. The worm searches the network for computers with vulnerabilities. There are different types of vulnerabilities. Some make it possible for a worm to execute malicious code directly on remote computers. The worm doesn't even have to send a copy of itself in a file. It can reside in runtime memory and propagate further to new hosts [6].

2.5. What are the differences between Trojans, viruses and worms?

A Trojan doesn't replicate itself like viruses and worms. The Trojan tries to fool users to make copies of it, and spread it to other users. The success of this tactic depends on how good its outer layer of useful or pretended functionality is. The Trojan program is directly executed by the user.

A virus executes itself and replicates itself by infecting program files on the computer. A virus in contrast to a Trojan, is not purposely executed by the user, it is indirectly executed when the user runs a program infected by the virus.

A worm doesn't replicate itself by infecting program files. The worm instead creates new files with a copy of itself, and then sends the new copy to other hosts, or it can replicate by executing itself directly on the remote host through some vulnerability on the host.

3. Propagation methods

Malicious software has used different methods of propagation during time. When the first viruses were written it wasn't common with computers connected to each other by networks. Users who wanted to exchange data or computer programs had to use removable media. (Removable media are transportable drives or disks which are easy to move from one computer to another.)

During this time viruses had to use removable media to propagate. The process looked like this. The virus had to infect programs or the boot sector on the removable media. When the media was moved to, and mounted on the other computer, the user had to do a step to finalize the propagation. This step could either be to execute a virus infected program on the media or to execute an infected boot sector on media [6].

As time went on new ways was developed to exchange information between computers. One of these was to use bulletin board systems. Users could now upload their files and download others from the system via modems. In this case the process of propagation starts when a user uploads an infected file to the BBS. The process is finalized when another user downloads the file and executes it [7].

Today computers are often connected to a local-area network and/or the Internet. The human intervention needed in the process of propagation of malicious software is getting smaller and smaller, because the process of propagation of files in general, between computers, is getting easier and more automated.

3.1. Propagation strategies used by worms

Worms are a group of malicious software that depends on network connectivity as the method for propagation. Worms are divided into three broad categories, in the paper *Recent Worms: A Survey and Trends*. The paper names the groups: E-mail (and other client application) worms, Windows file sharing worms and Traditional worms [8]. The paper *A Taxonomy of Computer Worms* defines three propagation carriers which worms use to spread from a computer to another. The first is called **self-carried** and describe a worm that "actively transmits itself as part of the infection process". The second carrier, **second channel** describe how some worms need to use "a secondary communication channel" to download additional worm code. The third, **embedded** describe how a worm "sends itself along as port of a normal communication channel" [9].

We now look at different communication strategies used by worms. How the strategies affect the speed of propagation and the stealth properties. The Stealth

property measures how good a worm can hide itself and pass undetected.

If a worm chooses the strategy to actively connect to other computers and initiate communication, the speed of the propagation might be high. As the speed is getting higher the stealth property might be reduced. This strategy is used when a worm wants to infect as many computers as possible before anti-virus companies detect it and distribute new signature files to their customers. If a worm chooses the opposite strategy, to passively wait for a normal connection to another computer, the speed of the propagation might be lower in comparison. The worm's stealth property is now higher and this may give it a longer lifetime, before detection and removal.

3.2. Propagation and human intervention

Some worms need no or very little human intervention to infect and propagate. Viruses and Trojan horses need more human intervention because they need to be transferred as a file from one computer to another.

Humans are motivated and fooled to help the propagation of malicious software by different social engineering techniques. Trojans pretend to offer the user something valuable, viruses depend on the value of the infected file. E-mail worms use techniques to trick the user to execute the worm with messages which "indicate urgency", "appeal to individuals' vanity" or "appeal to greed" [9].

A website called *The Register* had an article about why office workers help the propagation of malicious software. There are different reasons among the workers but they can be summarized as follows [10]:

- Not aware of basic virus prevention measures
- Too busy, they don't check their emails before opening them
- Believe that they have no part to play in preventing the spread of viruses
- No clue to how a virus-infected email might look like
- Too busy, they don't download anti-virus updates

4. Target strategies used by worms

For a worm to propagate it must discover which machines to infect. It can do this in several different ways. Scan based worms try to spread themselves by randomly testing IP addresses. Another way it uses to spread is through target lists. When using a target list, the creator of the worm will create or obtain a list of vulnerable hosts before releasing the worm. We will look closer at the target lists.

One of the problems for a worm that want to spread fast is that it grows exponentially. This means that it grows slow in the beginning. Using a target list would make the worm reach a high infection percentage faster in the beginning. It could do this with the help of "hit-list scanning". When a worm first infects a host it begins to scan down the first half of the target list. The next infected host is infected and gets the lower half of the target list. This process continues.

The hit-list can be obtained with several different techniques. Among others, [11] mentions these interesting:

4.1. Port scanning

The creator could port scan hosts during a long time to make a hit list. There is a risk that the scanning is discovered, but it is unlikely that it would lead to any serious actions taken against the scanner. The creator could also use the same methods as hackers use to avoid being detected, for example connection through proxy servers. Port scanning is an old method used by hackers to discover vulnerable hosts. Tools and information on port scanning are easily available on the Internet.

4.2. Distributed scanning

An attacker could use already compromised hosts as "zombie machines" making them scan for vulnerable machines. Since the machines are compromised the attacker doesn't have to worry as much of detection and could scan a lot more possible hosts.

4.3. Spiders

A spider is a computer program that looks at web pages and then visits the links of the webpage, and makes indexes of the pages visited. It goes around the "net" like a spider. This is the way search engines on the net works. If the worm infects web servers it can use spider methods to create a hit list.

4.4. Public surveys

If a security hole exists in a certain operating system or web server a worm creator could use public available listing. "http://www.netcraft.com"

4.5. Metaserver

A metaserver holds a list of possible machines to connect to for some applications. It is a match making service. If, for example, a person wants to play a com-

puter game with some other person over the Internet, he would use the server to see what other players there are to connect to. A metaserver worm would query a metaserver to discover new hosts to infect.

5. Infection methods

There are a number of different types of infection methods used by viruses and worms described in this section.

5.1. Executable File viruses

Here we describe the different methods used by file viruses to infect executable files.

5.1.1. Companion virus

A companion virus does not change the target file directly. It tricks the user into running the virus program instead of the real program by changing the name of the target file and/or the virus file. The W32/Parrot-A is an example of a companion virus. It renames all .EXE files in the /WINDOWS directory to .PRT and copies itself to the original filename. So when the user for example runs calc.exe in the /WINDOWS directory it will run the virus program instead of the calculator [12].

5.1.2. Link virus

A Link virus makes changes in the file system, so that the name of the file no longer points to the real file, but to the virus program.

5.1.3. Overwriting virus

This is a very primitive form of virus, which simply writes itself over the beginning of the infected file. The original program file gets damaged and when you try to run it all that happens is that the virus program is run. This is a very primitive method used by early viruses. Since it destroys the original program file it is easy to discover that you have been infected. This method is not often used anymore.

5.1.4. Parasitic virus

Parasitic viruses modify the contents of target files. A prepending virus places all of its code at the top of the original program so when you run the program, you first run the virus code and then the original program code. This way the user might not notice that he is infected by a virus. Another approach is to insert a jump operation at the beginning of the original program and then jump to the virus code which is inserted after the original program. After the virus program is run it returns to the beginning of the original program, and the original program is executed.

5.2. Memory resident virus

When some viruses are run they become resident in memory. They can actively search for files to infect or they can wait for another program to run and then infect it using any of the methods described above.

5.3. Macro and script viruses

The *Merriam-Webster Online Dictionary* defines the word **macro** as: “short for *macroinstruction,* a single computer instruction that stands for a sequence of operations” and defines the word **script** as “a plan of action” [13]. By these definitions you can draw the conclusion that a “macro is a script”.

What are macros and scripts used for and where? In the *help* for **Microsoft Word 2003** you get the following description about macros: “If you perform a task repeatedly in Microsoft Word, you can automate the task by using a macro. A macro is a series of Word commands and instructions that you group together as a single command to accomplish a task automatically.”

In other words, a macro is a programming language used inside documents to automate different tasks. If a macro language has functions for executing itself and replicating itself to other documents, this macro language can be used to write macro viruses. The infection process of a macro virus can look like this: The macro virus is automatically executed when an infected document is opened, and the virus replicates itself by copying the virus body to other documents on the computer.

5.4. Worm infection methods

Here we describe the different methods used by worms to infect a host. Since a worm by definition spreads itself over some network interface, we say that the worm infects a “host”.

5.4.1. E-mail worms

E-mail worms are programs that, when executed on a local system, take advantage of the user’s e-mail capabilities to send themselves to others. They try to trick the user into executing not trusted files. The first e-mail worms used local mail programs and/or mail APIs on the compromised machine to send copies of

themselves. Later worms have their own SMTP engine, which make them less dependent on the mail capabilities of the compromised machine [8]. Some e-mail worms search compromised computers for e-mail addresses, that it sends itself to.

5.4.2. P2P worms

The P2P is short for Peer-to-Peer. P2P worms spread through public file sharing systems such as Kazaa. A typical P2P worm copies itself to some of the shared folders used by the P2P program. It uses different filenames to trick users on the network to download it. Worms uses names of program files, pictures, songs etc.

P2P worms are not hard to write. They can be written in the Visual Basic Script language and doesn't require much technical or programming knowledge. Source code examples are available on the Internet.

The W32.HLL.Sanker is a recent P2P worm. Some of the names it uses are: hotmail_hack.exe, ea-games. Keygen.exe, girls.jpg.exe [14].

5.4.3. Windows file sharing worms

If a windows user wants to share files with other users on the Local Area Network he can do this by the peer-to-peer service that comes with Windows. The end user is responsible for setting the security properties, which can make the shared drive configured in an insecure way.

Many worms use the file sharing capability in addition to other methods. File sharing is often blocked outside of a Local Network by a firewall, but can be an effective method of spreading inside, for example, a corporate network. The windows file sharing worms is becoming more and more complex. The first ones just searched the neighbourhood network for unprotected drives and dropped files there. Later worms have had the ability to crack passwords which are poorly chosen.

5.4.4. Wireless worms

There haven't yet been any wireless worms that have made a large impact on the security community. Some experts believe this is because of the heterogeneous nature of the devices used; different hardware, different operating systems.

The Timofonica worm infected computers and send emails to an e-mail-to-GSM gateway, and therefore sending text messages to random cell phone numbers [15].

5.4.5. Traditional worms

This group of worms requires no human intervention to spread itself. It spreads by exploiting vulnerabilities in software. The software that the vulnerability is found in is often a operating systems. If a worm is going to have any impact and be widespread it has to have a lot of machines to attack and spread itself to. This is why a vulnerability in an operating system is serious; many users become potential targets.

A vulnerability can for example be a buffer overflow. It is interesting to note that most worms use known vulnerabilities. The problem is that many users aren't aware that they have to download a patch to make them not vulnerable. Long time after the security community knows of a problem it can still be exploited by a virus/worm.

Even security products can be attacked. The Witty worm [16] exploited a buffer overflow vulnerability in security products from ISS. The ISS products are supposed to parse incoming ICQ packets. The witty worm fooled them by pretending to be a ICQ packet and could therefore spread over the Internet.

The first internet worm, the Morris Worm, from 1998 was a traditional worm. It spread by exploiting vulnerabilities in the UNIX operating system.

5.4.6. Worms using several different methods

A worm can use several of the methods mentioned above. The Nimbda worm had five different ways of spreading itself: e-mail, windows file sharing, from web server to client browsing and two different ways to spread from a client browser to web server.

The way it spread from web servers to clients was to infect all web content files on compromised servers. This meant that a person surfing the net could unknowingly download a copy of the worm just by surfing the net. By exploiting a vulnerability in Internet Explorer some clients would automatically run the downloaded file and be infected [17].

6. Eluding methods

Malware use different ways to prevent detection and removal. Here we will explain some of the methods.

6.1. Disabling anti-virus software

Some malware disables anti-virus and firewall programs to not be detected. The `W32.Klez.H@mm` [18] worm, disables anti-virus scanners by stopping their processes. It also deletes their checksum database files.

6.2. Blocking web pages

A Version of the MyDoom worm [19] blocked users from accessing several web sites. Web sites included Microsoft update sites, and several anti-virus software sites. This was probably done to prevent users from getting information on how to remove the worm.

6.3. Encrypted viruses

Anti-virus scanners often use a special sequence of bytes, a signature, to detect viruses and worms. To avoid this virus writers began to encrypt their viruses. A typical encrypted virus has a virus decryption routine and an encrypted virus body. When the virus is run, the decryption routine first decrypts the virus and then the decrypted virus body is run. To encrypt the virus body the virus uses a different key every time, making the virus body different from infection to infection. Although the virus body is different the decryption routine remains the same, which sometimes makes it useful to use as a virus signature to search for [20].

6.4. Oligomorphic viruses

Since the encrypted virus has the same decryption routine from infection to infection it can be detected. An oligomorphic virus uses a different decryption routine from infection to infection. An oligomorphic virus can only create a finite number of different decryption routines. The Win95/Memorial virus could create 96 different decryption patterns. An oligomorphic virus could still be detected by an anti virus scanner searching for signatures, but it would take longer time [21].

6.5. Polymorphic viruses

A polymorphic virus is like an oligomorphic virus but it can create an *endless* number of different decryption routines. Like the encrypted virus it has a decryption routine and an encrypted virus body. But now the decryption routine is also encrypted, this is done by a mutation engine (a mutation engine "generates randomised decryption routines that change each time a virus infects a new program" [20]).

How is infection done? When an infected file is run the virus decrypt both the virus body and the mutation engine. A copy of the virus itself and the engine is made in memory. The virus then calls the mutation engine which creates a new decryption routine. This routine is used to encrypt the virus and engine in the RAM memory. And then the virus appends the new decryption routine and the encrypted virus and engine to a file. The first widespread polymorphic viruses were found 1991 [20]. Viruses and worms still use polymorphism to avoid detection. For example the widespread Bugbear worm had a variant which used polymorphism.

6.5.1. Generic decryption

Polymorphic viruses can be detected by using generic decryption. According to [20] generic decryption is based on three assumptions:

- "The body of a polymorphic virus is encrypted to avoid detection"
- "A polymorphic virus must decrypt before it can execute normally"
- "Once an infected program begins to execute, a polymorphic virus must immediately usurp control of the computer to decrypt the virus body, then yield control of the computer to the decrypted virus."

Based on this assumption a scanner can use generic decryption to detect polymorphic viruses. It does this by using a software virtual computer. When a program file is going to be scanned to see if it is infected the scanner loads the file into the virtual computer. In the virtual computer the file is run. Remember that this is in the virtual computer, so it can make no harm to the "real" computer. If the file is infected it will run its decryption routine first, which will decrypt the virus body. The scanner will (hopefully) detect suspicious virus-like behaviour and continue execution of the virus. The scanner can search the body for known virus signatures after the decryption routine is run. If the file is uninfected it will show no virus-like behaviour and the scanner will stop running the file in the virtual computer [20].

The problem with generic decryption is speed. It can take long time to decrypt the virus. How long can the scanner wait before it should begin to scan the next file. If it stops to early it might miss some virus. To remedy this situation the anti-virus companies came up with Heuristic-Based Generic Decryption [20].

6.5.2. Heuristic-Based Generic Decryption

Another method to detect polymorphic viruses is to use heuristic-based generic decryption. This is generic decryption with a set of rules, used to help scanners to know when to stop scanning and when to continue. If a file that is run makes some suspicious action then it might be a virus and the scanner should continue. For example a NOP instruction (telling the processor to do nothing) is behaviour which tells us that we might be dealing with a virus. On the other hand if the program

generates DOS interrupt it is unlikely that it is a virus. The scanning continues as long as the scanner is not sure that it is not a virus.

Heuristics can also be used to detect unknown viruses. If the level of virus-like code is above some previously defined threshold, the scanner will report a "possible infection" [22].

There are problems with heuristics, the biggest being that it may miss viruses. Virus authors will try to make their virus look like non-malicious programs. And the authors of the scanner will have to change their heuristic to answer that.

There have been suggestion of other methods involving Virus Cryptoanalysis [23], but when we wrote this we found no information about any anti-virus scanner using this method.

7. Case study of a virus

In this section we will describe what techniques a particular virus use for infection and stealth etc. We wanted to exemplify our definition of a virus. This was a challenging task because even if there are categories of malicious software, they are not mutually exclusive. For example, sometimes viruses act like worms, by propagating over networks [8].

We choose to study a virus from 1995 that follows the definitions of a virus very well. The virus belongs to the virus family Zhengxi. We pronounce the name **Zh-eng-xi** ("**j**-**ang**-**ch**-**i**", **j** in **j**ewel, **ang** in l**ang**uage, **ch** in **ch**urch and **i** in mach**i**ne). We will refer to Zhengxi as "the virus" in this case study.

The virus is a very complex DOS virus with stealth and polymorphic properties. It infects executables files, object files and archives in the ARJ, RAR and ZIP format. The archives are infected by appending a small COM file with the virus to the archive [25].

The work of analyzing the virus has been done by various anti-virus companies. We will summarize in the following sections what these companies have found out regarding this family of viruses. How they: install and hide themselves, infect files, are triggered, use anti-analyzing techniques. And finally who could be the author of the virus.

7.1. The installation process

The virus infects executable-, object- and archive files. An encrypted version of the virus is stored in these files. The start-up sequences differ a little bit between the file types, but they will all end up in a routine for the polymorphic decryption.

When the decryption of the encrypted version is done the virus body receives the control. The virus starts to look for signs telling it if the operating system DOS is present on the computer. If it is, the virus checks the system for more information. From this information the virus decides if it will continue the installation or terminate it.

The installation is terminated if: the date of the infected file is near the current date, Microsoft Windows is installed, an anti-virus scanner is running or the computer is booted from a floppy (drive A or B) [24].

The installation continues to allocate memory for a memory resistant copy of the virus. When the virus is loaded into memory it modifies the interrupt handler for 21h and 25h, so that the virus will intercept all calls to these interrupts. From now on the virus moves around in memory making it harder to be analyzed.

The installation is now finished and the virus lets the host program execute, in the special case where the installation was initiated from a COM-file the virus prints "Abnormal program termination" [25].

7.2. The stealth process

In the installation, the virus modifies the interrupt handler for 21h so that the virus will intercept calls to this interrupt. The interrupt 21h is called often because it has functions for file handling and memory management. When programs call these functions the virus will return modified results to hide the presence of the virus in memory [24].

To hide the presence in files which are opened for writing, the virus disinfects the files, with some exceptions. The virus has a list of programs which are allowed to open and write to infected files.

To hide the presence in files which are opened for reading, the virus decrypts the original file and returns it to the caller.

The virus hides itself also by returning modified file lengths for infected files and make it sure that programs cannot read outside this boundary [25].

7.3. The infection process

The virus randomly selects files to infect from the set of file accessed via interrupt 21h. The virus terminates the infection process if the file: is on a floppy (drive A or B), is bigger then available disc space, is dated near the current date or is not an executable, object or archive file.

The virus infects an executable file either by appending, inserting or by adding a file to a self-extracting file archive (ZIP etc).

The general idea behind the infection is to hide the virus code in the file. This is done by using polymorphic encryption. The virus encrypts the file header and the virus body and saves it to the end of the file. The

virus marks the file as infected by increasing the file size until file size modulus 157 equals 37 [24].

When the file is executed the virus body is decrypted and executed. Finally the virus decrypts the file header and hands over the control to the host file.

The virus infects by an inserting method when it finds C or Pascal subroutines in an executable file. The virus modifies a subroutine so it will execute the virus body. The virus can be inactive during long periods of time if the modified subroutine call is seldom executed [25].

When the virus starts the infection process of an archive file, it checks that the archive isn't already infected. If it is not infected it creates a COM-file containing random data, encrypted virus code and the decryption routine. This file is then added to the archive by the virus which has all functionality needed to add files to ARJ, RAR and ZIP archives [24].

7.4. The trigger and payload

When the virus finds a file in an archive packed with the no compression method it looks at the date of the archive. If the date has a year that equals to or are higher than 1996, the virus deletes all files and directories on all drives [25].

7.5. A difficult virus to analyze

Eugene Kaspersky tells in "Zhengxi: Saucerful of Secrets" how the virus was analyzed. He describes five techniques used by the virus author to make the analysis harder, these were [26]:

1. Use of polymorphism
2. Concealing of real functionality by using "hundreds of junk instructions" in the code
3. Use of different methods to access the same data in the code. "Not even good disassemblers could build the complete reference tables which are often so useful in analysis"
4. Use of "*hidden branches* where the destination address is concealed and may vary"
5. Use of checksums to "decide whether or not to infect a file". This makes it hard to figure out which files the virus is searching for. (It is easy to calculate checksums on data, but it is much harder to deduce from a checksum the set of data which generates the same checksum, and select possible candidates.)

7.6. Who was the author?

Who was the author of this complex virus? It is not publicly known who the author was. But by looking at how complex the code was, it's probably not a child. This conclusion was drawn in a document presented at the international Virus Bulletin Conference in 1996 by Sarah Gordon. She says "it is easy to see that Zhengxi is more of a programming exercise than a virus designed to spread. Its author must have known that the features which he was adding were just a display of skill. Clearly, it is not the work of a child, but of a programmer who could earn real money writing real programs in the real world" [27].

8. Case study of a Worm

In January 2003 the Sapphire worm, also known as SQLSlammer and W32.Slammer, began spreading on the Internet.

The worm exploited a vulnerability in the Microsoft SQL Server 2000. The vulnerability was discovered by NGSSoftware [16]. A vulnerability note was posted by CERT on 26th of July, 2003 [28]. The vulnerability was a stack buffer overflow. The MSSQL Server always listens on UDP port 1434. Clients can send a request of one byte to that port to discover how they should connect to the server. By using this stack based buffer overflow the worm executes code with the privileges that the Server had when it was started.

8.1. Background Stack Overflows

One common way for worms to spread is by exploiting some sort of overflow vulnerability.

In a classic article [29] the author "Aleph One" shows how by "smashing the stack" you can return from a routine to a random address. We will now give you a short explanation of how stack overflows works, which is needed for the understanding of the Sapphire worm.

In the programming language C a buffer is often represented by an array. A dynamic variable such as an array is allocated at runtime on the stack (Static variables are located at load time). When a buffer/array is overflowed it is filled with data outside of its bounds. This can happen when you use some of the standard C functions for copying and appending strings as for example strcopy(), strcat() and sprintf(). This is possible because there are no built-in bounds checking in C.

When a program calls a procedure it saves the return address on the stack. By overflowing the buffer that also is located on the stack a new return address is written over the old one. When the procedure returns it

looks at the return address and jumps there. By doing this the attacker gets control over the execution.

The attacker also needs some way to insert the code that he wants to be executed. This is simply done by putting the code in the buffer that will be overflowed.

8.2. The Sapphire stack overflow

The MSSQL Server always listens on UDP port 1434. When it receives a packet with the first byte set to 0x04 it takes the remaining part of the packet and tries to copy it to a stack based buffer. If a large number of bytes are appended to the packet the buffer is overflowed and the return address from the stack is overwritten. This gives the worm control of the execution, with the same rights as the MSSQL Server had when it was running.

8.3. Background Dynamic Link Libraries

The worm needs to use some pre-defined procedures to do its work. It needs to send packets and it needs some random number that it uses to generate IP addresses. It uses a DLL-file. To do this it uses the functions in the ws2_32.dll file that comes with the Windows operating system. Here is Webopedia's [30] definition of what a DLL is:

> Short for Dynamic Link Library, a library of executable functions or data that can be used by a Windows application. Typically, a DLL provides one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL. A static link remains constant during program execution while a dynamic link is created by the program as needed. DLLs can also contain just data. DLL files usually end with the extension .dll,.exe., drv, or .fon.
>
> A DLL can be used by several applications at the same time. Some DLLs are provided with the Windows operating system and available for any Windows application. Other DLLs are written for a particular application and are loaded with the application.

The procedures the worm needs are available in the library file ws2_32.dll, a file that contains the Windows Sockets API used by most Internet and network applications to handle network connections [31].

8.4. Background Import Address Table

Before the worm can use the procedures in ws2_32.dll it has to load the dll file into its memory. How does it do this? It uses the LoadLibraryA procedure. The LoadLibraryA procedure is a function that exists in the kernel32.dll file. The kernel32.dll is a file that is run as a Windows Kernel Process. This gives all windows programs access to its functions.

But how does the worm access the LoadLibraryA procedure in this process? It uses the Import Address Table of the sqlsort.dll library. Here is a definition of IAT [32]:

> Every win32 executable application has an Import Address Table (IAT) residing inside the program. The IAT is used as a lookup table when the application is calling a windows API function.
>
> There are many MS Windows operating systems, and they all have different addresses for their API functions, because of different structured DLLs. When an application starts, it has a list of all functions that aren't originally part of the application. These functions, called imports, are located in the operating systems DLLs, but the application doesn't know where. So before starting, it has to build the IAT table by finding the address of the API it wants to call.

"Calling a windows API function" is exactly what the worm does, so the IAT is useful here.

8.5. Sapphire code overview

This section is based on [33], [34] and [35], in which you will find the whole source code with comments.

Before we describe step by step what the Sapphire worm does we will show you a high level interpretation of the code. This code is from Symantec's Analysis [33] (We have added the two first steps: "Get control" and "Build packet"):

```
//1. Get control.
[get control]
//2. Build packet.
[build packet]
//3. Get handles to some libraries that
//   functions are imported from.
Ws2_32handle = LoadLibrary("ws2_32.dll");
Kernel32handle =LoadLibrary("kernel32.dll");
//4. Load GetTickCount(), and create a
//   random IP address with this value.
getprocAddress( kernel32handle, "GetTickCount");
IpAddress = GetTickCount();
//5. Socket setup
[Socket setup]
//6. Start infinite loop of generating a
//   new IP address, and attacking.
While (true) {
  IPAddress = generateRandom(IPAddress);
  Send Copy of Self through UDP
}
```

8.6. Sapphire description

This is a more detailed description of the code seen in Sapphire code overview.

8.6.1. Get control

The worm overwrites the return address and gets control of the processor.

8.6.2. Build packet

The first thing the worm does is to start building the packet that will be sent to other computers. It builds the packet on the stack. Some part of the worm is already on the stack. This part is marked [worm body] in the text below. The worm needs to rebuild its header section, because it can become corrupted.

First it loads the address of a "jmp esp" instruction. This is the address that will overwrite the real return address. It points to a "jmp esp" instruction in SQLsort.dll.

Then it loads a lot of garbage onto the stack. Remember that now the worm is building the packet that is going to be sent to the next victim. This garbage is needed so that the new return address overflows the exact location of the old return address, when the buffer later will be overflowed.

At the end of the garbage, the worm pushes a 0x04 byte onto the stack. This is required to be at the start of the malicious UDP Packet. The 0x04 byte tells the IIS server what kind of packet it should receive. By telling the server that it is a packet of "kind" 0x04 the server will go to the vulnerable code with the buffer overflow.

Now the stack looks like this:

[worm body]	Part that doesn't need rebuilding.
42B0C9DC	The return address.
GARBAGE	
GARBAGE	
0x04	The start of the packet

8.6.3. Get handles to some libraries that functions are imported from

Again, this is what the worm does in a high level language:

```
Ws2_32handle = LoadLibrary("ws2_32.dll");
Kernel32handle = LoadLibrary("kernel32.dll");
```

To do this the worm will first load some strings on to the stack that it will need during the remaining execution.

Now the stack looks like:

[wormbody]
0x42B0C9DC
GARBAGE
GARBAGE
0x04
'kernel32.dll'
'GetTickCount'
'ws2_32.dll'
'socket'
'sendto'

Now the worm uses the LoadLibrayA function to load the ws2_32.dll into memory. It saves the handle (Base address) to the file on the stack.

Then it pushes a string pointer for 'GetTickCount' onto the stack for later use.

Then it uses the LoadLibraryA function to load the kernel32.dll into memory and saves the handle (Base address) to the file on the stack.

The stack now looks like:

[wormbody]
0x42B0C9DC
GARBAGE
GARBAGE
0x04
'kernel.dll'
'GetTickCount'
'ws2_32.dll'
'socket'
'sendto'
[base address of ws2_32.dll]
[pointer to 'GetTickCount']
[base address of kernel32.dll]

8.6.4. Load GetTickCount(), and create a random IP address with this value.

This is what the worm does in this stage:

```
getprocAddress( kernel32handle, "GetTickCount");
IpAddress = GetTickCount();
```

First it loads the entry for GetProcAddress from the IAT in sqlsort.dll

Then it Calls the function GetProcAddress with the parameters kernel32_base and GetTickCount. Here it uses the pointer to the 'GetTickCount' string that was stored above. The GetProcAddress call returns the address to the function GetTickCount(). This address is used to call the GetTickCount. This function returns a random number which will be used as a seed for the worm's random number generator later. The worm saves this number on the stack. But before this number is pushed the worm pushes 8 zero bytes to the stack that will be used later to store an address.

The stack now looks like:

[wormbody]
0x42B0C9DC
GARBAGE
GARBAGE
0x04
'kernel.dll'
'GetTickCount'
'ws2_32.dll'
'socket'
'sendto'

```
[base address of ws2_32.dll]
0x00000000
[pseudo random seed]
```

8.6.5. Socket setup

```
A socket address of this type will be built:
Struct sockaddr_in {
    short   sin_family;   // Protocol type
    ushort  sin_port;     // Port number of socket
    struct  in_addr sin_addr;  // IP address
    char    sin_zero[8];       // unused
}

This is what the worm does:
// Use the type AF_INET, which means
// IP addressess to locate a computer
sin_family = 2;
// This is the port the worm sends the UDP
// packet to and also the port the server listens to
sin_port =1434;
getprocAddress( ws2_32-handle, "socket");
getprocAddress( ws2_32-handle,"sendto");
```

The worm first pushes the sin_family value and the sin_port value onto the stack. It then locates the address of the 'socket' API call with the GetProcAddr like above. Then it creates a UDP Socket with the socket API call and locates the sendtofunction like before.

8.6.6. Start infinite loop of generating a new IP address, and attacking.

```
This is what will be done:
While (true) {
IPAddress = generateRandom(IPAddress);
sendto(
  socket s,
  const char FAR *buf,
  int len,
  int flags,
 const struct sockaddr FAR * to,
 int tolen);
}
```

The target IP addresses are generated with a random seed and mathematical combinations.

The parameters:

S= the socket descriptor to the socket that was created in step 5.

buf=address on the stack where the packet that the worm have built begins. It is the address where the 0x04 byte is.

Len=376, the packet to be sent is 376 bytes.

Flags=0, no special behaviour by the UDP packet.

To=The address to the sockaddr_in structure created in step 5.

Tolen=10h, the structure is 16 bytes in length.

8.7. Comments

The Sapphire worm exploited a vulnerability in software from Microsoft. This was in January 2003.

During the writing of this paper a new worm emerged on the Internet. The Sasser worm began spreading on the 1 May 2004. See [36] for a good analysis.

We have no exact number on how many computers that have been infected by Sasser, but a few days after the outbreak numbers varied between 300 000 and a million infected hosts. Several companies have had their daily operations disturbed [37].

Sasser exploits a vulnerability in Microsoft Windows XP/2000. We think this demonstrates that worms exploiting vulnerabilities in widespread software will continue to be a big problem unless some action is taken to prevent this. We see two possible solutions to the problems: 1) Users becoming more security aware. 2) Software companies making safer software.

8.7.1. Users becoming more security aware

The Sapphire worm (and the Sasser worm) could be avoided by applying a patch from Microsoft. We think that the number of infected machines demonstrate that many users don't look for updates on a regular basis.

Another way to protect against worms is to use a firewall, which blocks ports often used by worms. Installing and managing firewalls can be very difficult and may therefore not a solution that fits the every home user.

8.7.2. Software companies making safer software.

Sapphire and Sasser would probably not exist today if the software had been more thoroughly tested for vulnerabilities before it was released.

9. Definition of an effective method

The definition of the word effective in *Collins Cobuild English Dictionary for Advanced Learners* [38] is: "Something that is effective works well and produces the results that were intended." We therefore define an effective method as "A method that works well and produces the results that were intended."

10. Effective viruses/worms methods

We define an effective virus/worm as a virus/worm that has effective methods for propagation, eluding and infection.

10.1. Effective propagation methods

By definition a virus doesn't actively spread itself to other computers like worms do. The virus propagates passively by the copying of virus infected files from one computer to another. We believe that a virus can propagate more effectively if it chooses to infect files with a big probability of being copied. For example if a computer is running file sharing programs it would be more effective to infect files in the shares rather than seldom used files that are not shared.

There are different types of worms. Some depend on the success of social-engineering techniques other on vulnerabilities found on computers. Out conclusion is that these worms depend on "passive humans". For example "passive humans" do not check their e-mails for viruses before opening them. They don't apply the latest software updates, thus letting vulnerabilities exist in the system.

The most effective propagation method depends on the target. If the target is users that are hard to fool then it might be better to look for vulnerabilities in their computers. On the other hand if it is easier to fool the users than finding vulnerabilities, this would be the more effective method.

10.2. Effective infection methods

In the case study of the Zhengxi virus we saw that it infects different types of files: executables, compressed executables, object files and archives. We think this virus is a good example on how an effective infection method should work. The strategy of infecting files in a computers file share is more likely to succeed if the infection methods support many types of file formats. A virus missing archive support can not infect archive files in a file share thus less effective compared to viruses like Zhengxi!

10.3. Effective eluding methods

Some malicious software use eluding methods to hide its presence from anti-virus software/users. The Zhengxi virus used polymorphic encryption to hide itself in files. While the virus was running in memory it used stealth techniques to conceal its memory-usage and the increased file length in infected files. The virus even disinfected in some cases infected files in order to be undetected.

The virus tries to elude the anti-virus companies when they analyze it. This is done by using techniques as "hidden branches" etc. It is probably harder to come up with a countermeasure to a virus if it is hard to analyze. This gives the virus more time to propagate before countermeasures can be taken.

11. Effective countermeasures

We define an effective countermeasure as a countermeasure that has effective methods against propagation, eluding and infection.

11.1. Effective methods against propagation

One method to propagate malicious software was to infect file shares on computers. We think that one effective countermeasure to this threat would be to prevent write access to the files in the shares. We have never heard about situation when users want to modify files in share folders. How often do users patch MP3-, movie- or program files that they share? No, they don't but malicious software would probably do it!

Nowadays malware often use the Internet to propagate. A good countermeasure against this threat would be to scan and block malware with a firewall. The firewall finds malware by matching the attachments against signature files. But what about if the attachment is a malware hidden inside an encrypted archive? There is no risk whatsoever to get infected by an encrypted archive if you don't know the unlock key. The user must decrypt the archive with the unlock key and then execute the malware. If the unlock key is a part of the e-mail message to the user, the firewall can try to use parts of the message body to decrypt the attachment. If it finds the unlock key it may scan the attachment for malware.

11.2. Effective methods against infection

We believe that a good method against infection is to run programs in a sandbox, in which you specify which files individual programs are allowed to read/write or execute. If a certain program tries to break one of these rules the users should be notified about this.

Worms use vulnerabilities such as buffer overflows. An effective countermeasure against this threat would be to install memory guards either in software or in hardware to prevent programs to write outside buffers.

11.3. Effective eluding methods

The eluding method used by the Zhengxi virus was to redirect some operating system calls to it. The virus modified the results from these calls to hide the virus existence. We think that a good countermeasure against this method would be to protect the operating system against modifications.

12. References

[1] BBC News, “Why people write computer viruses”, http://news.bbc.co.uk/go/pr/fr/1/hi/technology/3172967.stm, 2004-03-24.

[2] Micha F. Lindemans, ”Trojan Horse”, http://www.pantheon.org/articles/t/trojan_horse.html, 2004-04-14.

[3] Trend Micro, “Virus Primer”, http://www.trendmicro.com/en/security/general/virus/overview.htm, 2004-03-24.

[4] McAfee, “Virus Glossary”, http://us.mcafee.com/VirusInfo/VIL/glossary_app.asp, 2004-03-24.

[5] Symantec, “What is the difference between viruses, worms, and Trojans?”, http://service1.symantec.com/SUPPORT/nav.nsf/pfdocs/1999041209131106, 2004-02-10.

[6] Princeton University, “How Computer Viruses Spread”, http://www.princeton.edu/~protect/BasicConceptsAndTips/Viruses/HowComputerVirusesSpread, 2004-04-06.

[7] Wikipedia, “Computer virus”, http://en.wikipedia.org/wiki/Computer_virus, 2004-04-06.

[8] Darrell M. Kienzle, Matthew C. Elder, “Recent Worms: A Survey and Trends”, http://portal.acm.org/citation.cfm?id=948189&dl=ACM&coll=portal, 2004-04-06

[9] Weaver, Paxson, Staniford, Cunningham, “A Taxonomy of Computer Worms”, http://www.cs.berkeley.edu/~nweaver/papers/taxonomy.pdf, 2004-04-06

[10] The Register, “Clueless office workers help spread computer viruses”, http://www.theregister.co.uk/2004/02/06/clueless_office_workers_help_spread/, 2004-04-06.

[11] "How to 0wn the Internet in Your Spare Time", http://www.icir.org/vern/papers/cdc-usenix-sec02/cdc.web.pdf, 2004-04-07

12 Sofos virus analysis: W32/Parrot-A, http://www.sophos.com/virusinfo/analyses/w32parrota.html, 2004-04-06.

[13] Merriam-Webster Online Dictionary, http://www.m-w.com/, 2004-04-06.

[14] Symantec W32.HLLW.Sanker worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.sanker.html, 2004-04-06.

[15] ZDNet, “Got a bug in your ear”, http://zdnet.com.com/2100-11-521320.html?legacy=zdnn, 2004-04-06.

[16] NGSSoftware Insight Security Research Advisory, http://www.nextgenss.com/advisories/mssql-udp.txt, 2004-04-06.

[17] CERT Advisory CA-2001-26 Nimda worm, http://www.cert.org/advisories/CA-2001-26.html, 2004-04-06.

[18] Symantec W32.Klez.H@mm worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.klez.h@mm.html, 2004-04-06.

[19] Sofos virus analysis: W32/MyDoom-B, http://www.sophos.com/virusinfo/analyses/w32mydoomb.html, 2004-04-06.

[20] Symantec, “Understanding and Managing Polymorphic viruses”, http://securityresponse.symantec.com/avcenter/reference/striker.pdf, 2004-04-06.

[21] Péter Ször, Peter Ferrie, “Hunting for Metamorphic”, http://www.peterszor.com/metamorp.pdf, 2004-04-07

[22] Network Associates, “Advanced virus detection Scan Engine and DATs”, http://www.bytware.com/press/scan_engine.pdf, 2004-04-06.

[23] Mircea Ciubotariu, Virus Bulletin November 2003, “Virus Cryptoanalysis”, http://www.virusbtn.com/magazine/archives/200311/cryptoanalysis.xml, 2004-04-06.

[24] NOD32, ”Virus Encyclopedia”, http://ve.nod32.ch/viruses/z/zhengi.php, 2004-04-07.

[25] Kaspersky Lab, ”Virus Encyclopedia”, http://www.viruslist.com/eng/viruslist.html?id=2638&printmode=1, 2004-04-27.

[26] Virus Bulletin, Kaspersky, ”Zhengxi: Saucerful of Secrets”,

http://www.virusbtn.com/magazine/archives/pdf/1996/199604.PDF,
2004-04-07.

[27] IBM, Gordon, ”The Generic Virus Writer II”,
http://www.research.ibm.com/antivirus/SciPapers/Gordon/GVWII.html#NEWAGE,
2004-04-07.

[28] US-CERT Vulnerability Note VU#484891,
http://www.kb.cert.org/vuls/id/484891,
2004-04-06.

[29] “Smashing The Stack For Fun And Profit”,
http://www.shmoo.com/phrack/Phrack49/p49-14,
2004-04-06.

[30] Webopedia Computer Dictionary,
http://www.webopedia.com/TERM/D/DLL.html,
2004-04-06.

[31] LIUtilities WinTasks DLL Library,
http://www.liutilities.com/products/wintaskspro/dlllibrary/winsock/,
2004-04-06.

[32] The Free Dictionary,
http://encyclopedia.thefreedictionary.com/Import%20Address%20Table,
2004-04-06.

[33] Symantec SQLExp SQL Server Worm Analysis,
http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf,
2004-04-07.

[34] Matthew Murphy, “Analysis of Sapphire SQL Worm”,
http://www.techie.hopto.org/sqlworm.html,
2004-04-07.

[35] SAPPHIRE WORM CODE DISASSEMBLED,
http://www.eeye.com/html/Research/Flash/sapphire.txt,
2004-04-07.

[36] Sasser worm analysis,
http//www.lurhq.com/sasser.html,
2004-05-06.

[37] New victims for Sasser net worm,
http://news.bbc.co.uk/2/hi/technology/3683553.stm,
2004-05-06.

[38] Collins COBUILD English Dictionary for Advanced Learners, HarperCollins Publishers 2001.