Johanna Kraken

Analysis of malware - the Morris Worm

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology (in English)

Bachelor’s Thesis

28. May 2019

Author Title Number of Pages Date	Johanna Kraken Analysis of malware - the Morris Worm 45 pages 28. May 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology (in English)
Professional Major	Mobile Solutions
Supervisor	Kimmo Saurén

It has always been necessary to secure your personal information. This has become all the more necessary since the Internet has become obsolete as people have begun to steal these and other data through the internet or by other means.

This thesis deals with the different types of malware and their history. There are three different kinds that are further discussed here. The first one is the virus and the second is the worm. The largest different between them is that the Virus needs a user to activate and the worm can spread itself. The last one is the Trojan. That is a program that hides inside a legal program.

In addition, the *Morris Worm* is analyzed and its consequences are shown. The Worm was programmed by Robert Tappan Morris, but did not want to cause any damage. The Worm sent a message to an IP-address and cracked passwords of the computer on which it was. The cracked passwords were only saved in the code of the virus and not sent to someone. Mr Morris made a mistake that led to multiple infected computers that decelerated and to non-responsive servers. Mr. Morris was sentenced to do 400 hours of community service and to pay over 10.050 US Dollar. In addition, he got three years of probation.

The thesis also discusses how to protect yourself from malware. The best way to protect yourself from malware is to follow three rules. First, you should always have your antimalware software active and up to date. Second, you should only open files whose origin you trust. Third, if you get an email but you do not know or trust the sender, you should never open the attachment.

Keywords	malware, cyber security, *Morris Worm*

Contents

LIST OF FIGURES

List of Abbreviations

VBS	Visual Basic Script. It is a scripting language developed by Microsoft.
ARPANET	Advanced Research Projects Agency Network. It was an US military computer network which is the internet founded on.
UDP	User Datagram Protocol. Computer applications can send messages with UDP to other hosts on an Internet Protocol (IP) network.
TCP	Transmission Control Protocol. Computer applications can send messages with TCP to other hosts on an Internet Protocol (IP) network.
DDoS	Distributed denial of service. It is a kind of cyber-attack that makes the specific internet service unreachable.

1 Introduction

At the latest after the hacking of the accounts of almost 50 million Facebook users became public in September 2018, everyone knew, whether young or old, that not even a company as large as Facebook is safe from hacker’s attacks. According to the company, the attackers had exploited two weak spots in the function with which Facebook members can view their profile from the perspective of other users. These errors were caused by a bug in Facebook's video-upload program. The point in time of the attack is unclear, but it must have been after the launch of the software feature in June 2017. The hackers got so-called access token with which they were able to get access to the personal accounts (Frenkel, Sept. 29, 2018). That shows that everything that is programmed by a human is not 100% secure because it is normal that human make mistakes.

However, not only companies are attacked by hackers. Also the average consumer can be a target. Most of the people that have been hacked are not aware of it or just do not think about it. For example, everybody is aware of procedures like the Grandparent Scam (The victims get a call from their ’grandchild’ who is in trouble and needs money, but nobody should know who is in trouble.). Nevertheless some people do not think that when you are using the computer and surf the internet or open an email, hackers can get access to the computer in case the computer has no sufficient virus protection program.

To increase awareness about malware, this thesis focuses on malware and their functionality.

1.1 Structure

In this thesis the different kinds of malware are analysed and the differences are shown between viruses, worms, trojans and other malwares. The thesis also covers the history of the malware. In addition, it gives an overview over the cost of cybercrime today and short outlook on the future (for more details, see chapter 2).

Chapter 3 discusses the *Morris Worm*. It explains what it is and what caused the spreading. Also, the main structure and the consequences of the *Morris worm* are presented.

Chapter 4 explains how to protect yourself from malware in different ways, while chapter 5 provides the conclusion.

1.2 Objective of the Thesis

The objective of this thesis is to analyse a malware - the *Morris Worm* from 1988. This worm changed the view of the malware. From this point in history, malware was divided into two types – the virus and the worm – because the latter malware type behave differently from the others before. It spreads by itself and do not need a user to activate it (Nazario, 2004).

In addition, the thesis provides information about the different kinds of malware and their history. It also explains how to protect oneself from malware and how to deal with a malware that is already on your computer.

2 Malware

Information about the different kinds of malware and malware in general are provided in this chapter. Also, the history of malware and cybercrime is treated.

2.1 Types of Malware

When people are surfing the internet, they tend not to worry about the topic of internet security, and they accidentally download a malware when they least expect it . However, it is important to understand different types of malware and what they are capable of performing (Roger, 2018).

'Malware' is an abbreviation of the term 'malicious software'. That refers to code which has been created deliberately harmful. The malware can manipulate, delete, move and/or copy data or disrupt the workflow of the computer or network. In addition, malware also can aim damaging of the physical hardware of the systems (Cisco, 2018).

Well-kown examples are viruses, worms and trojans. Each of these malware types has its own peculiarities and malice and is fundamentally different from another. Often it is difficult to clearly assign a given malware to a class. For example, some malware belong both to the virus and the worm category because it contains properties of both (Cisco, 2018).

2.1.1 Virus

Virus is the most well-kown malware, but it is not the most common one. In the year 2017 viruses were 26% of the malware under Windows (for more details, see Figure 1).

“Viruses are the only type of malware that ‘infects’ other files” (Roger, 2018). It is very hard to eliminate viruses because they are hidden inside and executed from a legitimate program most of the time. A virus also alters the behavior of the infected computers. The virus can serve to destroy valuable data or cause unrepairable damages (Roger, 2018).

The virus propagates to another computer in which it leaves infections behind by leaving a copy of itself behind. Viruses can cause damage to files or software and can lead to critical conditions for the system. It is also possible that a system is infected, but the virus is not active or can spread to other systems. Only when the host program is executed (e.g. an email attachment is opened), the virus hidden inside is executed. To spread the virus, the host program, with a copy of the virus hiding inside, has to be transferred to a different system by using any kind of transportation (e.g. email attachment and file sharing) (Cisco, 2018; Nazario, 2004).

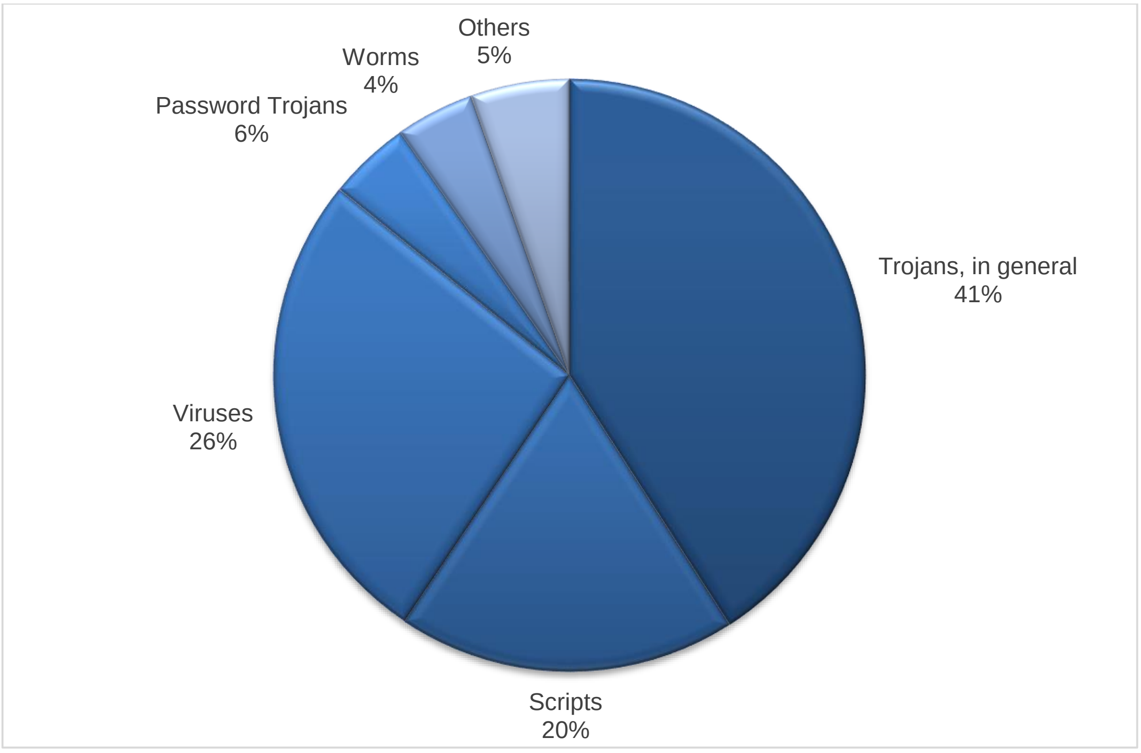

Figure 1 Distribution of Malware under Windows 2017 (The Independent IT-Security Institute, 2018)

One famous example of a virus is the *Michelangelo virus*. It is well-kown because it was reported on by the media in 1991. The destructive virus was programmed to override sectors of the hard drive as soon as the infected computer was booting on the 6th of March which is the birthday of the renaissance artist Michelangelo. The effects were not as bad as expected in the end. And because the virus spent most of its time inactive it was imaginable that an infected computer could be for years undetected— as long as it was not booted on that date, while infected (Skoudis & Zeltser, 2003, pp. 38-40).

Another example is the *CIH virus* in 1999. Sometimes it is also called *Chernobyl* because it took place on the 13th anniversary of the Chernobyl's disaster. *CIH* makes the system not bootable by rewriting parts of the flash memory of the BIOS, which leads to an infinite loop. This is causing the system to hang and forcing the user to hard-reset the system. Some old systems did not allow the reset by users or admin. That is the reason why they had to contact their manufacturers to get new chips for the motherboard (Skoudis & Zeltser, 2003, pp. 477-481).

2.1.2 Worm

The worm is the oldest malware, but it is not commonly used. In the year 2017 the worm was only 4% of the malwares under Windows (for more details, see Figure 1). The term 'worm' was coined by the book Shockwave Rider by John Brunner originally published in 1975 (Nazario, 2004).

What makes the worm so dangerous is the fact that it replicates itself and spreads from system to system. It will not connect to the infected computer operating system. In addition, there is no need for an end user to spread it (it could be activated, for example, by opening an inactive program as the virus), which makes the worm very effective and dangerous. One way to spread a worm is by sending emails, which was the reason why it became popular in the late 1990s (Roger, 2018; Nazario, 2004).

Moreover, a worm spreads itself by using an unprotected part in the target system. Once in the system, it moves through the system unassisted by taking "advantage of file-transport or information-transport features" (Cisco, 2018). Another feature of a worm is, that it can communicate with other programs (Nazario, 2004).

The *Code Red* worm appeared in July 2001. It is a binary-only and memory resident worm. The weak point of this worm was a buffer overflow problem. This problem is caused when a system receives more information than it can handle; this causes an override of the adjacent memory. The *Code Red* worm was programmed to execute a

DDoS attack on the White House by having all the infected computers call the website of the White House at the same time to overload the server. *Code Red* had two successors: *Code Red 2* and *Code Red* ||. The *Code Red 2* was nearly the same as the original. The difference from the original was the technique which created the backdoor on the host system but used the same weak spot as the original (Skoudis & Zeltser, 2003, pp. 19, 76-82; Nazario, 2004).

The *Love Bug* or *Love Letter* is another example of a worm. This VBS worm was spread via Outlook email in May 2000 just like the *Melissa worm* 1999. The subject line was "ILOVEYOU". The user had to open the executable attachments to get infected. The VBS worm was used to spread the email distribution lists which can contain thousands of new possible victims and which the hackers got without the intervention of the user due to a large number of software flaws on the email servers. A large number of companies disconnected themselves from the internet for a short time until the worst part was over. Other companies were one or two days off- line (Skoudis & Zeltser, 2003, pp. 19, 46, 76-82; Nazario, 2004; Helfrich, 2018).

2.1.3 Trojan

The Trojan is the most common malware. In the year 2017 the general trojan was 41% of the malware under Windows; in addition, there is another 6% of trojans which are programmed for getting password from the victim. The distribution of malware can be seen in Figure 1.

The Trojan got its name from the ancient Greek tale of the Trojan War (Trojan Horse). Like in the tale, the Trojan sneaks into the computer of the victim while pretending it is a legal program which the user would like to use. In Figure 2 you can see one way to get infected by a Trojan. It is the malware which is the most common because it is hard to detect for firewalls or conventional defenses and easy to program. The most popular way to get infected by a Trojan is by downloading a program that pretend to be a fake antivirus program (Roger, 2018; Helfrich, 2018).

There are many ways in which the malware can attack the host, after being activated. It can confuse the user with opening and/or closing windows or modifying the desktop by adding/deleting/stealing files or data. Trojans also can open infected files, for example, by a virus or worm so that they can spread and do whatever they were programmed for. Backdoors are also created by trojans so that an invader gets access to the system (Cisco, 2018).

There can be a confusion with the terms backdoor and Trojan or Trojan horse, but there is a large difference between them. The backdoors are the system that gives access to a program. The Trojan is a program that pretends to be a legal program. Certainly, there are programs that are both backdoors and trojans at the same time (Skoudis & Zeltser, 2003, pp. 188-189).

In contrast to viruses and worms, this kind of malware do not multiply itself or other files. It needs the help of the user to be spread through the system (Cisco, 2018).

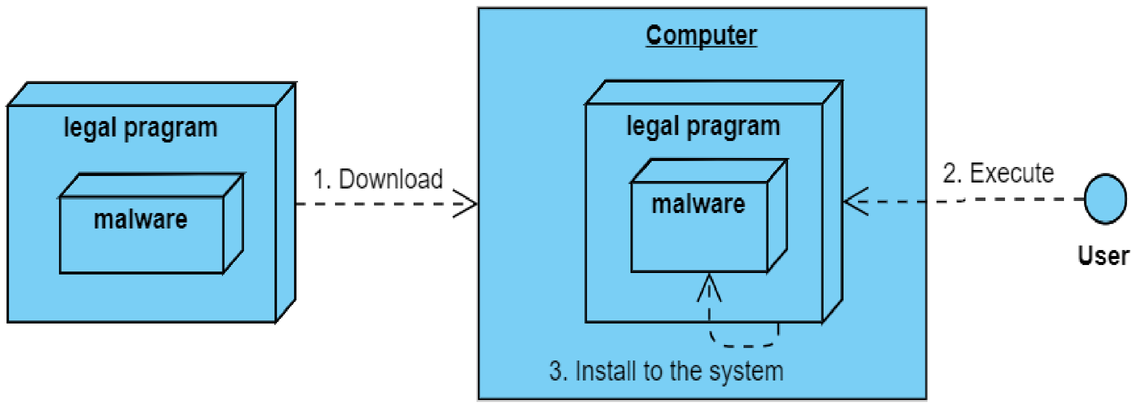

Figure 2 One way to get infected by a Trojan

A technique that conceals data inside of executables file is called Hydan (2003). It uses polymorphic coding techniques to hide the data. Polymorphic code means that there can be different codes, but the result is the same (Skoudis & Zeltser, 2003, pp. 19, 294-295).

One of the first Trojan programs was *ANIMAL* written in 1975 by John Walker. The program was written to copy itself to removable media when it is connected to the system by masquerading as a game (Helfrich, 2018).

Another Trojan family named *AntiVirus 2008* was created in 2008. This family was programmed to masquerade as a free antivirus program that locates a large number of malware on every system. It actually provides malware among other things by switching off the qualified anti-virus system (Helfrich, 2018).

2.1.4 Others

Other malware is, for example, Spyware. Parents can use it to watch the computers of their children for example. But it can also be used by criminals, as the name suggests, to spy on the victim’s computer without their noticing it (Skoudis & Zeltser, 2003, pp. 147-149).

Bots are another example. They can have different application features. They can automate tasks and provide services and information which is usually executed by a human as the derivation of the word ‘robot’ suggests. It can be used as a chat bot that helps in chat conversations (e.g. to collect information). But it can also be used to gain control over a computer or to connect the computer with a network of infected device or ‘botnet’ with which hackers can launch large-scale attacks on other victims. Advanced botnets can also use internet of things (IOT) devices (e.g. in a Smart Home) to raise a larger automated attack. Besides the ability to spread itself like a worm, the bot has the ability to record keystrokes, pass on information’s (e.g. passwords) and gain access to the host by opening backdoors. Furthermore, the bots are known to take advantage of backdoors unlocked by worms and viruses to get full control of the network (Cisco, 2018).

2.2 History of Malware

The history of malware goes back to the 1940s. The Hungarian Computer scientist John von Neumann (1903-1957) thought about a mathematical automata that can reproduce itself (the term ‘virus’ was not used at that time) and presented the manufacturing of this program in the year 1951. It was later published as the "Theory of self-reproducing automata". At that time no one thought that virus programs could be harmful (Kaspersky Lab IT Encyclopedia, n.d. -a).

Computers were rare at that time, which suggests that malware was also rare. As soon as more and more people got computers, more people tried to do new activities with it - including the development of malware (Kaspersky Lab IT Encyclopedia, n.d. -a).

2.2.1 1970s

One of the first viruses was the *Creeper virus* in the early 1970s. It was using the ARPANET. The Creeper did not course damage to data. The only effect was the displaying of the sentence “I'm the creeper: catch me if you can”. A short time after the *Creeper*, the Reaper was launched. That is a virus that was programmed to delete the *Creeper* from infected systems. It could be called an Antivirus, but nobody knows who created the *Reaper* (the creator of the Creeper or other people) or with which intension (e.g. to undo a mistake or just to delete the *Creeper*) (Kaspersky Lab IT Encyclopedia, n.d. -b).

After the novel Shockwave Rider was published in 1975 the Xerox Palo Alto Research Center (PARC) intended to develop the first worm application. The Shockwave Rider contained the statement that worms should be considered as weapons. The infected computers were connected to a network to exchange information or later to execute calculations which otherwise take a long time to be calculated with only one computer. But like in the novel the inventors build accidentally a ‘vampire’ virus that crashed the computers in the network and denied them normal operation. In order to get rid of the ‘vampire’ worm, it had to be completely eradicated by all systems (Nazario, 2004; Helfrich, 2018).

In the same year the Trojan program *ANIMAL* was written by Johan Walker (for more details, see section 2.1.3. Trojan).

2.2.2 1980s

At the beginning of the 1980s the *Apple II* computer system was widespread used. It was not only a virus but also a game system. These computers were at least infected by three self-contained viruses. One of these programs was *Elk Cloner*, which was spread by booting the computer from an infected floppy disk which copied the virus to the computer. It was created by high school junior student Rich Skrenta. It worked in the way that in

case the computer was infected, and the disk was not, then the virus was copied to the floppy. At that time the floppy disks were used to store the operating system and often shared by friends, which benefited the spreading. However, this 'virus' did not spead very strongly and was therefore never seen as a virus (Kaspersky Lab IT Encyclopedia, n.d - c.; Skoudis & Zeltser, 2003, pp. 16-17, 29-30; Helfrich, 2018).
This program output is shown in Figure 3:

```
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!
```

Figure 3 Sentence that the cloner virus showed (Skoudis & Zeltser, 2003, pp. 29)

On November 10th, 1983 there was history written in the computer safety. Len Eidelmen was the first who characterized self-replicating computer programs as a ‘virus’ at a seminar on computer safety at Lehigh University. On year later the term ‘computer virus’ was born. Mr. Eidelmen defined the term as a program that can ‘infect’ others by copying themselves to the program (Kaspersky Lab IT Encyclopedia, n.d. -c).

1985 the Trojan *Gotcha* was written. It was programed to delete data on the system and show the text ‘Arf, arf, Gotcha’ by masquerading as a fun program that should show ASCII-ART characters on the display (Helfrich, 2018).

The *Brain virus*, which began its mischief in Pakistan in 1986, was the work of two brothers, Basit and Amjad Farooq Alvi. The brothers owned a company in Pakistan called Brain Computer Services (Skoudis & Zeltser, 2003, pp. 17, 30-31; Helfrich, 2018).

It was not difficult to discover by whom the virus was launched because the virus code contained the text which is shown in Figure 4.

"Welcome to the Dungeon (c) 1986 Basit * Amjad (pvt)
Ltd. BRAIN COMPUTER SERVICES 730 NIZAB BLOCK ALLAMA IQBAL
TOWN LAHORE-PAKISTAN PHONE :430791,443248,280530.
Beware of this VIRUS.... Contact us for vaccination............ !!"

Figure 4 Sentence that the Brain virus showed (Skoudis & Zeltser, 2003, pp. 30)

In 1988 several malwares were detected. One of them was the *Jerusalem virus*, which affected computers all over the world on Friday, May 13th. The virus searched and infected all .COM- und .EXE-files and caused a general slowdown of the computer. The infected files were growing by each infection around 1800 bytes. It was enjoying greater popularity - probably because it was quite simply programmed and would be an ideal 'foundation stone' for someone experienced in assembler programming to create new viruses infecting COM and EXE files (Kaspersky Lab IT Encyclopedia, n.d. -d).

One other virus was the *Morris worm* 1988 named after Robert Tappan Morris. The author was a student at Cornell University. It was spread via the internet and gained as the first worm media attention. It was written as a research project but due to a mistake in the spreading mechanism it became a rapidly spreading worm. After the infection the worm was sending a 1-byte packet to the University of California. It was designed to track how the worm spreads. It does no harm nor creates backdoors (Nazario, 2004; Helfrich, 2018; Indiana University Bloomington, n.d.).

One year later a modification of the *Jerusalem virus* appeared called the Datacrime and FuManchu which was threatening. It operated from October 13th through December 31st to irrevocably destroy data (Kaspersky Lab IT Encyclopedia, n.d. -e).

2.2.3 1990s

In 1990, the year in which the first polymorphic virus, *V2Px* (the *Chameleon family*), was found. These viruses were re-encrypted with each infection, which made the detection of the virus or the development of anti-virus software considerably more difficult. Mark Washburn, the author of *Chameleon*, combined two different viruses to create the virus. The re-encryption feature made the antivirus programs of that time unusable. But it did not took long until the antivirus experts created new algorithm which were able to detect the polymorphic virus (Kaspersky Lab IT Encyclopedia, n.d -f.; Helfrich, 2018).

The *Michelangelo virus* from 1991 is one of the most known viruses in the 1990s (for more details, see section 2.1.1 Virus).

In 1995 a worm spread for one month but did not caused real damage. It was the predecessor of many of this kind and it was on the hit list of the antivirus program. It was called *'Concept'*. In the first half the Digital Equipment Corporation (DEC) accidentally spread the worm but it was quickly detected and stopped (Kaspersky Lab IT Encyclopedia, n.d.-g; Helfrich, 2018).

One of the most devastating malwares in the 1990s was the *Melissa* electronic-mail worm in 1999. It was designed to send malicious emails with the copy of the worm included by each infection. These emails were sent to 50 people in each address book. To slow the spreading of Melissa down it was necessary to change the configuration in the software of the email program (Nazario, 2004):

2.2.4 2000s Until now

At the beginning of the new century the *Love Bug* and *Code Red* appeared (for more details, see section 2.1.2. Worm). The *Love Bug* was considered the most damaging worm of its time, infecting millions of computers worldwide. Another worm that showed up in 2001 was the *Nimda worm*. It invaded IIS server with the same vulnerability as *Code Red* but with different attack techniques. It was using mails like *Code Red*, the opening of Windows networking file shares and the uploading of an exploit to an infected website (Nazario, 2004).

In January 2003 a worm was released in South Korea. The *SQL Slammer* worm was programmed not only for computer systems but also for a wide range of systems for example ATMs (crashed) and emergency services like 911, fire or police services (no longer available). It spread very fast, but it produced so much traffic by trying to spread in other countries that the internet in South Korea broke down. That is a reason why this worm was not as damaging as it could have been. A more evil one would have reduced the bandwidth to be able to increase. The *SQL Slammer* used UDP for communication, whether the most other worms used TCP. The advantage of this is, when sending a message with UDP it is not necessary to wait for an answer and send an ACK back (three-way handshake). After sending the message, it can go over to send the next message to the next system, as demonstrated in Figure 5 (Skoudis & Zeltser, 2003, pp. 14, 19, 79, 93, 102-104).

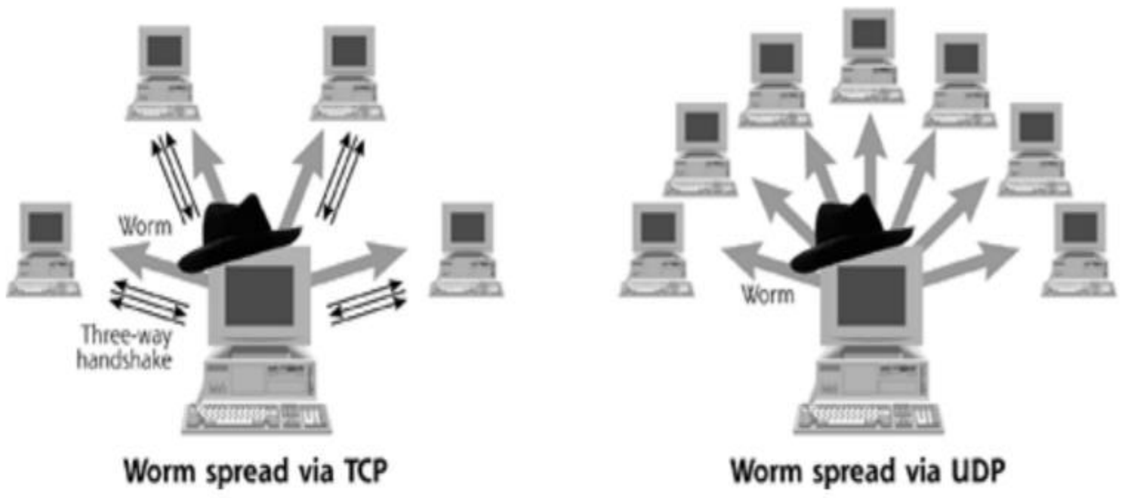

Figure 5 UDP and TCP mechanism (Skoudis & Zeltser, 2003, p. 104)

The first mobile virus '*Cabri'* appeared in 2004. It spread by using open Bluetooth connections to search for others and transmit the virus code to it. That is the reason why it spread so fast in other countries. This virus's aims were Nokia phones under Symbian (Helfrich, 2018; Kaspersky Lab IT Encyclopedia, n.d. -h).

In the year 2008 the trojan *‘AntiVirus 2008’* family appeared which helped malware to get access to the system unlike the name suggests (for more details, see section 2.1.3 Trojan).

In the year 2016 a trojan masqueraded as a login website of a bank. Once a login trial went wrong, it guided the victim to the actual bank website. That is the reason why the most victims did not notice that their data was stolen (Helfrich, 2018).

2.3 Cybercrime - Today and in the Future

Not only in the history was malware used to attack companies and single persons. The year 2017 saw some of the most disastrous attacks. In the US the University of Maryland’s Clark School discovered that one third of all American companies already were attacked by cyber-hackers. Every 39 seconds a computer which is connected to the internet is under attack by a hacker. But you can apply this statistic to other countries as well. Cyber hackers do not care about the location of their targets. They use the World Wide Web; thus, they do not have to be in the near in order to attack. They even do not have to be on the same continent. That is one reason why everyone should be prepared to be the target of criminals (Alvarez Technology Group Inc, n.d.).

Ransomware is a specific kind of malware that take away the control of your system and gives it usually only back if you pay ransom money. During the time the attackers have control over your system they deny you access to files and/or programs on your system. But even if you paid the ransom money, it would not be guaranteed that you get the full access back (sometimes they still have a way in the system, for example, with an backdoor or something similar) or that everything is restored like it was before (Alvarez Technology Group Inc, n.d.).

A study of Osterman Research uncovers that ransomware’s downtime effect lead to huge losses to companies in 2017. A great number of companies turn off their systems for hours – sometimes even longer than 100 hours. It also discovered that ransomware was the most usually used for attacks (Alvarez Technology Group Inc, n.d.).

The attacks on small to medium-size companies grew in the year 2017. In one year, the cyber-attacks on this group grew up to around 6%. This caused damages to the companies. They lost not only employees but also customers data because of cyber theft. This information is confirmed by another study of the Small Business Trends which discovered that this group is targeted more and more because they tend to find the security holes late. The main reason of that might be that they often do not have the resources and manpower for a good cybersecurity. The study found that the cyber-attacks percentage increased from 15% to 43% of the total amount of the attacks between 2011 and 2015 (Alvarez Technology Group Inc, n.d.).

The biggest threat to the security of a company are the employees, according to the Ponemon Institute's 2017 study. Careless employees cause up to 54% of the security holes. The cybercriminals often aim at the employees with malevolent emails and websites. This threat is easy reduced by giving the employees some rules like always logout, do not login from an unsecured Wi-Fi and of cause hold back your passwords (Alvarez Technology Group Inc, n.d.).

The global damage caused by ransomware attacks amount to five billion USD in 2017. That is 15 times more than two years earlier in 2015. The damage is not only the downtime but also the connected penalty and losses of productivity. Also, the ransom money is included in the cost but just is a tiny fragment of the total cost (Alvarez Technology Group Inc, n.d.).

By 2021 the yearly damage caused by cybercrime is predicted to be six trillion USD. One reason is that it is getting more profitable, which makes it attractive to future and current criminals. Another reason is that it is challenging to capture the cyber-criminals and sentence them, unlike the drug dealers (Alvarez Technology Group Inc, n.d.).

3 Analyses

This chapter discusses the *Morris Worm*. The reason for and the consequences of the spreading of the worm are addressed. Also, the worm itself is analysed.

3.1 *Morris Worm* – Overview of the Facts

The *Morris Worm* was programmed by Robert Tappan Morris, a graduate student at Cornell University. Unlike other malware, it was designed as an experiment and not to generate any real trouble. On November 2nd 1988 Morris uploaded the worm to the internet from a MIT computer to conceal that it came from his university. Morris mainly was directed at one operating system, Unix (Indiana University Bloomington, n.d.; Boettger, 2000).

Even without steeling data from the systems, it caused massive damage. The infected systems did not only spread the worm fast but also slowed down the system. The worm was supposed to be just once on the computer. If it discovered that it is for the second time, it should stop. But it did not. The worm was still running. Because of that it was running multiple times on the computer. That was the reason why the computer's memories, drives, and processors were slowing down and stopped working. The only way of not getting infected was to disconnect from the internet completely (Boettger, 2000; Indiana University Bloomington, n.d.).

Morris realized fast that his program was spreading much faster than it was supposed to do. He sent an email to a friend at the Harvard University in order to get help. At last, an anonymous report was sent to Harvard by his friend in which they explained how to delete the worm and to protect themselves against another infection. Nevertheless, it was too late to stop the worm. Among the infected websites were already many important ones like those of universities or governments. The expected expenses for the deleting of each infected system were calculated to be between 200 and more than 53,000 US Dollar per worm on each system. The total damage cost varied but it is around 98 million US Dollar partly because of man-hours which were needed to find a solution to the problem and deal with the damage (Indiana University Bloomington, n.d.; Boettger, 2000).

Within 12 hours a team at Berkley discovered a method to slow down the spreading. Another team at Purdue released a different technique, as well as a team from MIT that also worked on a solution. However, these solutions were too late since many websites were already down and had no connection to the internet anymore. When the first solution worked, nearly 6000 – around 10% of the total number of computers in the world – were infected. They were also several computers which were disconnected from the internet before getting infected or to prevent the spread from continuing but the number of them is uncertain and were not counted to the 6000 victims (Indiana University Bloomington, n.d.; Boettger, 2000).

This chaos calmed down after a few days. Now everyone wanted to know who the person was who designed it and the New York Times named Morris as the author (even though it was not official verified yet). But there were many indications pointing towards Robert Morris. Robert Morris turned himself in, motivated by his father – UNIX OS co-author and chief scientist at NSA’s National Computer Security Center. That fact was considered by the court. He was sentenced for violating the Computer Fraud and Abuse Act (Act 18) of the 1986 US Federal Law. He had to do 400 hours of community service and to pay over 10.050 US Dollar in fine. In addition, he got three years of probation. The sentence is controversial. For those who worked on fixing the damage, the worm had caused this penalty seemed too light. On the other hand, other people thought that the punishment is either good or even too hard. Two years after the release of the worm he appealed but it was refused three months afterwards (Indiana University Bloomington, n.d.; Boettger, 2000; Kaspersky Lab, 2003).

The *Morris Worm* was called a virus for a long time. But there is a large difference between a virus and a worm. A worm can spread to other computers by itself. A virus needs to be spread by something or someone to other computers. It can be an USB-stick or another software program. But they have one thing in common. They cause problems for everybody who is involved. It does not matter if it is the user or the support team of the computer. Sometimes the problems are small. But sometimes they can be programmed to cause huge damage - even if it was not intended (Boettger, 2000).

3.2 Actions of the *Morris Worm*

To understand what the *Morris Worm* did, you first have to understand what it did NOT do:

- Did NOT manipulate, delete, move and/or copy files
- Did NOT send or save the decrypted passwords (except for its own list for favorites passwords which it passed on to new worms)
- Did NOT try to capture superuser privileges or make use of privileges
- Did NOT saved copies of itself or other programs on the system which is executed later
- Did NOT attack systems which were not connected to the internet
- Did NOT cause martial damage at the system
- Did NOT use other ways to spread than the internet (e.g. disk)
- Did NOT manipulate other programs (e.g. to spread itself)
- Did NOT use the user action (e.g. for activating like a virus)
- Did NOT install trojans or create backdoors (because it must replicate itself before it is discovered)

After that is clear, the question is now what it did do. The intension of the programmer was that the worm was supposed to do nothing notable. It was supposed to spread itself as fast as possible without getting noticed from the user of the system. If the program had worked as intentioned, it had been a very small, ongoing process on the computers.

However, the program did not work like that because there were still a few bugs inside the code. The result was that the processes, which were on their own very small and did not take much processor time, were a burden on the systems as more and more copies of the *Morris Worm* infected the same systems. The system’s slowed down more and more with each copy of the worm that infected the system.

The Worm spread itself over the internet by taking advantage of vulnerabilities like modern malwares today. The worm used three different vulnerabilities. The first one is the Finger and Sendmail bug. The implementation of that in Unix-based systems permit the remote code performance. The next vulnerabilities will be only used if the first one is not fruitful. The worm tried to use the rsh (remote shell). It is normally utilized for remote control. But to use the rsh, the worm needs a username and password which it got by using a trial and error method. The worm tried out if an encoded word is equal to the encoded password. If not it will try the next one. At that time is was not so important to users to have a strong password so even the system administrator selected, for example, the username (forward or backward) or a simple word as a password sometimes (Kaspersky Lab, 2003).

As soon as the worm infected a system, it did everything to hide itself. For instance, it renamed to processes name, removed temporary files and encode itself in the memory. As one of the next steps the worm checked if already a copy exists on the system. If it found a copy, it should destroy itself. But in one of seven cases the worm did not perform the check. This is the reason why this sometimes led to the DDoS effect (Kaspersky Lab, 2003).

3.3 Analysis

The code can be divided into different main functions – mainloop(), checkother(), cracksome(), other_sleep() and report_breakin(). The main structure contains these functions amongst others. You can see the main structure in the Figure 6.

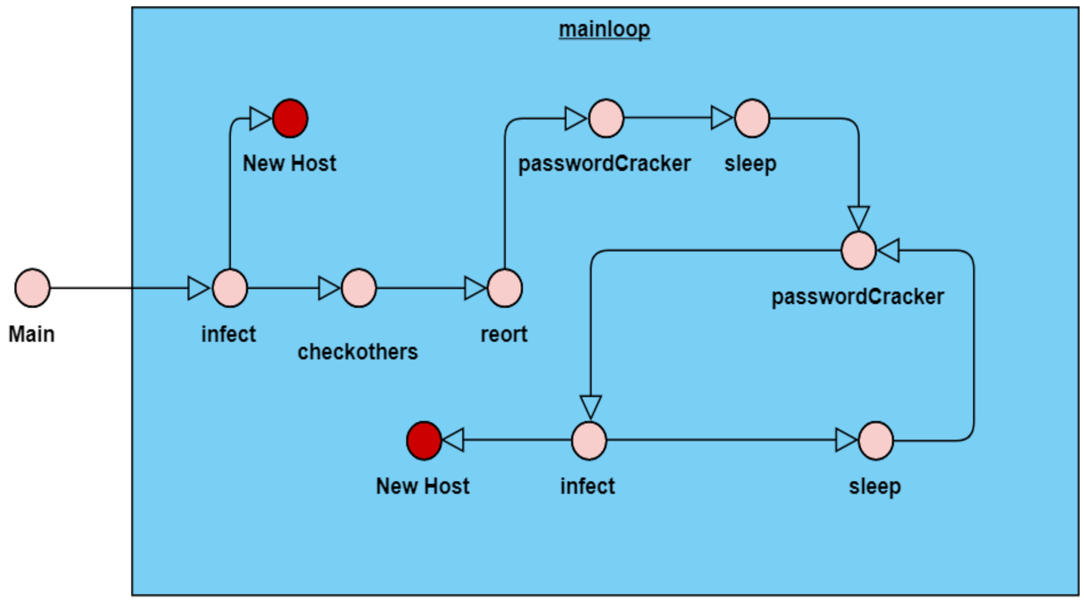

Figure 6 Main structure of the *Morris Worm*

3.3.1 Main Loop – mainloop()

The mainloop()-function can be dived into two parts. The first part is just called once. The second part is a loop. You can also see the pseudo-code of this function in the Figure 7. This function is located in the file named worm.c.

The first part is a series of functions that are called. You can see this as well in Figure 6 as in the Figure 7. In this first part those seconds in the variable ‘key’ will be saved that have passed since 1st January 1970. It is also called a random number generator with the variable named ‘key’.

Then the worm will infect some hosts by invoking the hg() and the hl(). The function ha() will be called next, if one or both of the beforehand called functions are not able to infect any hosts.

Next, the function checkother() is called to check if the other worms are already running on the system. After that the worm sends a single byte to "128.32.137.13" to report the successful infection of the system by invoking the function report_breakin().

Then it will try to crack insufficient passwords of user accounts on the system the first time by calling the function cracksome(). Next and last in this part the function other_sleep()-function with the number 30 is called.

The second part is the while-loop. It will be executed as long as a new created child process does not return to the parent or have more possible passwords to check.

The first function in this part is called the cracksome() to crack passwords as in the first part. Next, there will be a new process created. If the child process does return to the parent, the parent process will exit.

The next functions will be called the hg(), the hi() and the ha(). Those will try to infect new hosts. If one or a combination of the functions cannot infect a new host the function is called hl().

Then, the function other_sleep() is called the parameter 120. It will be sleeping and waiting until another worm is contacting this worm as in the first part.

After that, it will check with an if-function if 12 or more hours have passed by. This will be calculated by those seconds which have passed since 1st January 1970 and subtracted by those which were saved in the variable key from the beginning of the function. If the result is larger than the result of 60*60*12, the function h_clen() is called which resets the host table.

The next if-function which is the last part of the mainloop()-function will check if the variable named ‘pleasequit’ is null which is added in the checkothers() and the answer_other() and the variable named ‘nextw’ is greater than zero, which is added in the try_words(). If these conditions are given, the loop will be exited.

```
mainloop() {
   seed the random number generator with the time
   attack hosts
   checkother();
   sending a single byte to "128.32.137.13"
   cracksome();              // Crack some passwords
   other_sleep(30);          // Sleep, waiting for another worm to contact me

   while (1) {
      cracksome();           // Crack some passwords
      create new process and if return exit(0)
      attack hosts
      other_sleep(120);      // Sleep, waiting for another worm to contact me.
      if (12 hours passed by)
         reset hosts table
      if (pleasequit && nextw > 0)
        exit(0);
    }
}
```

Figure 7 ’C’ pseudo-code of the mainloop() function

3.3.2 Check if the Worm is Already Running – checkother()

The checkother() function used for checking if the worm is already running on the system in another process. This function is located in the file named ‘hs.c’.

At first it creates a random number and divides it by seven. If the remainder is three, it returns to the mainloop(). This means that in one out of every seven cases the worm becomes virtually immortal (if it passes this function there is a possibility that the variable named ‘pleasequit’ will be added by one which eventually exit the while-loop).

Afterwards the worm produces a new socket and checks if it can reach the port 23357 of its own system. If the socket establishes successfully the connection, they exchange random numbers to confirm that on the other side of the connection is another *Morris Worm*. If the transmitted number plus the received number was an odd number, the variable named ‘pleasequit’ is added by one. After that, the socket will be closed.

When five seconds have passed, a new socket on the same port will be opened to listen to other worms. In a case of error during any phase, the function will close the socket and return to the main function without trying to open a new socket. The socket is saved in the variable named ‘other_fd’ in case an error has occurred.

3.3.3 Report Back – report_breakin()

In this function the worm will report the successful break-in to the system to the computer with the IP-number ‘128.32.137.13’. This function is located in the file named ‘worm.c’. You can also see the program code in Figure 8.

This function is rather simple and is only 21 lines long. First it checks if a random generated number divided by 15 has the remainder seven. If the condition is given the function returns to the mainloop()-function without doing nothing similar to the checkother()-function.

Then it will create a new socket which send one byte to the IP-number ‘128.32.137.13’ (probably Morris IP-address which he used to follow the spreading of the worm). It will send the byte to the port 11357. If the socket creation failed, the function returns to the mainloop(). After that, it closes the socket.

```
static report_breakin(arg1, arg2)
{
    int s;
    struct sockaddr_in sin;
    char msg;

    if (7 != random() % 15)
                    return;

    bzero(&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = REPORT_PORT;
    sin.sin_addr.s_addr = inet_addr(XS("128.32.137.13"));

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0)
                    return;
    if (sendto(s, &msg, 1, 0, &sin, sizeof(sin)))
                    ;
    close(s);
}
```

Figure 8 Program code of the report_breakin() function (Martini, 2014)

3.3.4 Password Cracker – cracksome()

In the cracksome() function the worm will crack some passwords of the user account as the name suggests. This function is located in the file named ‘cracksome.c’. You can also see the program code in Figure 9.

This function is very simple. It just has a switch-case branching. This branching is a way to check if specific cases are true. If a case is given (e.g. the variable which is checked is a string named ‘hallo’) the function which is written under the case is executed.

In this function the variable which is checked is named ‘cmode’. The variable is an integer. The associated cases are zero, one, two and three. In every case, it is called a different function and after it has finished the cracksome()-function returned to the mainloop() .

```
cracksome()
{
    switch (cmode){
    case 0:
                    strat_0();
                    return;
    case 1:
                    strat_1();
                    return;
    case 2:
                    try_words();
                    return;
    case 3:
                    dict_words();
                    return;
    }
}
```

Figure 9 Program code of the cracksome() function (Martini, 2014)

3.3.4.1 Case 0

If the variable named ‘cmode’ is zero, the function strat_0() will be executed which is located in the file named ‘cracksome.c’. The function task is to find new hosts and encrypted passwords to crack.

At first the worm reads through two files named ‘/etc/hosts.equiv’ and ‘/.rhosts’ if it finds them. The first file “contains a list of trusted hosts for a remote system, one per line” without a password (Oracle Corporation and/or its affiliates, 2010). The other file does not contain a list of users but a list of host-user combinations. This list is normally used for granting the permission to a user to get remoted access from a specific host (Oracle Corporation and/or its affiliates, 2010).

The worm is opening these files to search for new host if they are present. A host that is found that way is checked whether it has the array called ‘o48’ and if it has at the position zero not a zero. If the conditions are met, the host variable named ‘flag’ will be used the bitwise OR with eight. If the conditions are not met, they will continue with the next host without modifying the variable.

After that the function setpwent() is used to open the ‘/passwd’ file . A loop was performed as long as passwords could be read with the function getpwent(). The function other_sleep() is called with the parameter 0 every 10th entry. Next, the file ‘.forward’ is opened for each user in the home directory of that user and read the hostnames therein. Not only the hostname is added to the host list, but also the encrypted password, the home directory, and the GECOS field are added to each user in the list named ‘user_list’.

After all passwords have been read, the file ‘/passwd’ is close by calling the function endpwent(), the variable named ‘cmode’ is set to one and the list ‘user_list’ is saved into a global variable named ‘x27f2c’. Then it returned to the cracksome()-function.

3.3.4.2 Case 1

If the variable named ‘cmode’ is zero, the function strat_1() will be executed which is located in the file named ‘cracksome.c’. The function task is to crack the really simple passwords.

The function contains for-loop. It loops through a block of code a number of times. It also contains three statements which decide the number of time the code will be executed. The first statement will be executed one time before the loop is starting. The second specifies the requirements for the execution of the block of code. The third will be executed every time when the block of code is executed.

The for-loop here has two conditions. First, it will only be executed 50 times. Secondly, it checks if the variable ‘x27f2c’ has successor which is saved in the variable named ‘next’.

To check a password the function, try_passwd() with two parameters is used. The first parameter was the user variable ‘x27f2c’ and the second one was the encrypted password. To encrypt the password the function XS() with a parameter, which is the possible password, is called. The parameter of the function try_passwd() is checked if the user variable named ‘passwd’ and the second parameter was the same. If that is the case, it saved that it could crack the password, it called the function attack_user() with the first parameter of the try_passwd() function and returned one. If that is not the case, it returns zero. If the function returns one, the strat_1() function continued with the next user.

The function attack_user () was only called from the try_passwd(). This function used the found password to break into this user account. The function opened the files ‘/.forward’ and ‘/.rhosts’ with the home directory of this user.

First, it tried if the user has even set a password. It called the function try_passwd() with the variable named ‘x27f2c’ and an encrypted empty string as parameters. Then it would try the username, username+username and different combinations of the GECOS field (e.g. the second value in the GECOS string) as possible passwords.

After the for-loop ended, the variable named ‘cmode’ is set to two if the variable named ‘x27f2c’ is zero.

3.3.4.3 Case 2

If the variable named ‘cmode’ is zero, the function try_words() will be executed; that is located in the file named ‘cracksome.c’. The task of the function is to try words as a possible password which were saved in an array named ‘wds’.

In the front of the function the array named ‘wds’ with a small dictionary of words and a global variable named ‘nextw’ with zero were initialized. The author probably thought of the small dictionary of words himself as a good chance for passwords. The arrays first position is only set with an object which contains the words.

At first the worm checked if the array that is at the position of the value of the variable ‘nextw’ is zero. If the condition is given the variable named ‘cmode’ will be added by one and the function returns to the cracksome()-function.

Next, the worm checks if the variable named ‘nextw’ is zero. If it is zero, the amount of the words in the array named ‘wds’ is saved in the variable i by using a for-loop. Then the array is encrypted by calling the function permute() with the parameter variables named ‘wds’ and the number one and the size of the array in the first position.

Afterwards there will be three for-loops called. The first one is going through the array named ‘wds’ in first position and checks if it is not equal to '\0'. If this is the case, the encrypted word at that position is using the bitwise AND with 0x7f. The second for-loop is going through the list named ‘x27f28’ and is calling the function try_passwd() with the parameters the actual ‘x27f28’ variable (the actual user) and the first position of the array ‘wds’ (the array of the possible, encrypted passwords). The last called for-loop is again going through the array ‘wds’ and is using the bitwise OR with 0x80.

At last the variable named ‘nextw’ is added by one. Then it returns to the cracksome()-function.

3.3.4.4 Case 3

If the variable named ‘cmode’ is zero, the function dicts_words() will be executed; the function is located in the file named ‘cracksome.c’. The function task is to try words as a possible password which were saved in a huge file on the computer.

Finally, the worm is opening the file ‘/usr/dict/words’ which contains a dictionary on nearly every system, which is a huge amount of words, and it is saved in the variable named ‘x27f30’. If the variable ‘x27f30’ is null after the opening of the file, the function returns to the cracksome()-function.

Next, the worm read a line from the file and saved it into the variable named ‘buf’. If the reading returns zero, the variable named ‘cmode’ will be added by one and the function returns to the cracksome()-function.

Afterwards two for-loops are executed. Both are going through the array named ‘x27f28’ and are executed every time the function try_passwd() with the parameter of the current user and the variable named ‘buf’. Between the execution of the for-loops it is checked if the variable ‘buf’ contained at the first position only uppercase characters. If that is the case, the function returns to the cracksome()-function. If that is not the case the variable ‘buf’ at the first position is convert to lowercase and then is the second for-loop executed with the modified array.

After the last for-loop is finished, the function returns to the cracksome()-function.

3.3.5 Sleep and Wait – other_sleep()

This function is located in the file named ‘hs.c’ and it is responsible for a number of tasks. Overall you could think that the name of the function would describe the main task of the function very good. But the main task of it is to detect other worms in the system.

The function is invoked with an integer as a parameter named ‘how_long’. This number is used later on. After that, it checks if the creation of the socket which is saved in the global variable named ‘other_fd’ and was first used in the function checkothers() has failed - the variable is negative. If the condition is given it will check if the given parameter is not zero. If it is not zero, the function will sleep the amount of seconds specified by the how_long variable. In either way the function returned to the main_loop()-function without doing anything.

Afterwards there is a do-while-loop. A do-while-loop first executes a block of codes and checks if the condition is given in the end. If the condition is granted the block is executed again. If it is not given the code after the block is executed. In this case, the loop is running as long as the variable named ‘how_long’ is larger than zero. If the variable how_long is smaller or equal zero, the function returns to where it left.

It will first check once again if the variable other_fd in the loop is smaller than or equal zero. If this is the case, the function is going back to the main_loop() right away.

After a timeout variable with the variable named ‘how_long’ as seconds used in a timeout variable is created; it checks if the variable ‘how_long’ is not zero. If it is not zero, the past seconds since 1st January 1970 is saved in the variable named ‘time1’. Then the function select is called and saved in the variable named ‘nfds’. The function is called with other_fd+1, the address of variable readmask, zero, zero and the address of variable timeout as parameters. The result is that the variable other_fd is saved in variable readmask after the bits have been shifted to the left by one place, but other_fd has not been modified.

The select-function checks if the variable readmask has become available for reading. The variable timeout defined the upper limit for the time that has passed before the function returned.

Lastly three if-queries are executed. The first checks if the variable named ‘nfds’ is below zero. If this is the case the function was sleeping for one second. The second query checks if the variable ‘readmask’ is not zero. If this is the case the function answer_other() is called. The last one checked if the variable ‘how_long’ is not zero. If this is the case the seconds which have passed since January 1st 1970 are saved in the variable ‘time2’. In the variable ‘how_long’ the subtraction is saved that contains how_long minus the variables time1 and time2. Finally, the function returns to the function from which it was called.

If the return value of the select-function specified that the input was pending for the variable named ‘other_fd’, it means there was currently another worm on the current machine. The function answer_other() is the opposite of the function checkother(). The function is trying to connect to the other worm and to exchange numbers and it is task is to verify the identity. Next it is checked if the return really came from the same computer (form the IP-address "127.0.0.1") If the addition of the two exchange numbers is an odd number the connection to the other worm is closed, the variable named ‘other_fd’ is set to minus one and the variable named ‘pleasequit’ is added by one. After that the socket will be closed. Finally, the function returns to the other_sleep().

3.3.6 Infect - hg(), hl(), ha() and hi()

The first infection before the loop in the mainloop-function consist of three function called - hg(), hl() and if both were returning zero ha(). The infection in the mainloop() has two different ways. The first one consist of two function calls - hg() and hl() - and if the return value of both is zero ha() is executed (for more details, see chapter 3.3.1). The second way consist of three function calls - hg(), hl(), ha() and if the return value of this three is zero hi() is executed (for more details, see chapter 3.3.1). These functions are located in the file named ‘hs.c’ and have the overall task to infect other hosts.

3.3.6.1 hg()

This function is located in the file named ‘hs.c’ and is only called from the mainloop-function. It has the task to infect gateway machines. You can also see the program code in Figure 10.

At first it is called the function rt_init(). That function initializes a list of gateways every time it is called again.

Next, a for-loop is used to go through the gateway array. For each gateway it is calling the function h_addr2host() with the gateway and the number one as parameters. The return value is saved in a variable named ’host’. That variable is used as a parameter for the function try_rsh_and_mail(). If the return value of the function is true the hg() return to the mainloop() with the value one. If it is not the case it continues with the next gateway.

After the for-loop ends, the function returns to the mainloop() with the value zero.

```
hg()
{
    struct hst *host;
    int i;

    rt_init();

    for (i = 0; i < ngateways; i++) {
          host = h_addr2host(gateways[i], 1);
          if (try_rsh_and_mail(host))
                return 1;
    }
    return 0;
}
```

Figure 10 Program code of the hg() function (Martini, 2014)

3.3.6.2 hl()

The hl() function is located in the file named ‘hs.c’ and is only called from the mainloop-function. It has the task to infect a host which at any position of hosts array named ‘o48’ is not equal zero. You can also see the program code in Figure 11.

First, a for-loop is used to go through a host’s array named ’o48’ of maximum length of six which is saved in the variable named ’me’. For each position that is not zero it is calling the function hi_84() with a Binary AND Operation of the value of the current position and the function return value of the function netmaskfor() with the value of the current position as the parameter. If the result is equal zero, the function returns to the mainloop() with the value one. If the result is not equal zero, it continues with the next position in the array.

After the for-loop ended the function returned to the mainloop() with the value zero.

```
hl()
{
    int i;

    for (i = 0; i < 6; i++) {
                    if (me->o48[i] == 0)
                        break;
                    if (hi_84(me->o48[i] & netmaskfor(me->o48[i])) != 0)
                        return 1;
    }
    return 0;
}
```

Figure 11 Program code of the hl() function (Martini, 2014)

3.3.6.3 ha()

The ha() function is located in the file named ‘hs.c’ and is only called from the mainloop-function. It has the task to infect hosts on remote networks.

First, it is checked if the global variable named ‘ngateways’ is smaller than one is. If it is smaller than one, the function rt_init() is called. Next, a variable j is set to zero.

Then a for-loop is executed. The loop is going through the global gateways array. For each array position the function h_addr2host() is executed with the current gateway and the number one as parameters. After that the return value of the function is saved in the local variable host. Next, another for-loop is executed. The loop is going through an array named ‘o48’ of host variable. Then it is checked if the current position of the array is zero. If it is zero, the loop continues with the next position. If the return value of the function, try_telnet_p() with the current position of the array as parameter is zero the loop is continuing with the next position as well. After both checks the current value of the array is saved in the j position of the local array named ‘l416’. The next step is that the variable j is added by one.

Next, the local array named ‘l416’ is encrypted by calling the function permute() with the parameter variables named ‘l416’ and the number one and the size of the array at the first position.

Afterwards, a for-loop is executed again. This time it is going through the array named ‘l416’. It is checking for each position in the array if the return value of function named ‘hi_84’ is one. This function is executed with the parameter the Binary AND Operation of the current position in the l416 array and the return value of the function named ‘net-maskfor’ .The function netmaskfor() had the current position in the l416 array as a parameter. If this is the case, the function returns to the mainloop() with the value one. If it does not return one, it continues with the next position in the array.

After the for-loop ended the function returned to the mainloop() with the value zero.

3.3.6.4 hi()

The hi() function is located in the file named ‘hs.c’ and is only called from the mainloop-function. It has the task to infect hosts in the host list which has the variable named ‘flag’ modified in the function strat_0() (for more information see chapter 3.3.4.1.). You can also see the program code in Figure 12.

First, it is used a for-loop to go through the global host list. For each host two conditions were check. The first is a Binary AND Operation of the current host’s variable named ’flag’ and 0x08 is not equal zero. The second condition is given if the function try_rsh_and_mail() with the current host as parameter return value is not equal zero. If both conditions are given the funtion returns to the mainloop() with the value one. If not it continues with the next host.

After the for-loop ended the function returned to the mainloop() with the value zero.

```
hi()
{
    struct hst *host;

    for (host = hosts; host; host = host->next )
            if ((host->flag & 0x08 != 0) && (try_rsh_and_mail(host) != 0))
                    return 1;
    return 0;
}
```

Figure 12 Program code of the hi() function (Martini, 2014)

3.4 Consequence

It is hard to have a shut down of approximately 10 % of the computers without any changes or consequences that are drawn.

At that time the users of the computers were most of the time universities or governments. And they were using insecure passwords as the *Morris Worm* has shown. It is probably true that a large number of people have the same or similar insecure passwords today as well. But at that time the necessily of strong passwords was not public knowledge. If stronger passwords had been used 1988 the worm would not have been able to use one of the first three cases of the cracksome() function (Boettger, 2000).

Furthermore, the awareness or concern of the data security in general was not very strong. Much more, there was an open trust between everyone which was using the internet back then. The general thought was that data is for everyone. But exactly that was what the *Morris Worm* took advantage of. For example the access to certain files like '/usr/dict/words', '/etc/hosts.equiv' and '/.rhosts' should be only granted to those who really need it (Boettger, 2000).

In addition to that, the most companies or establishments did not have trained computer security professionals with them working at that time. That would have been maybe caused less damage by the *Morris Worm*. Also, the network security was shown to be incapable of defending from such attacks. Today there are much more of them who help companies to prevent this kind from attacks. Furthermore, if 10% of the computers would get infected by a worm like the *Morris Worm* today and get disconnected from the internet or get shut down. It could be destructive for the companies (Boettger, 2000).

Moreover, the worm was a reason why CERT, National Computer Security Center and other organizations were created. The tasks of this organization are to prevent cases like this or even worst ones to happen (Boettger, 2000).

4 How to Protect Yourself From Malware

In this chapter the question ‘How to protect yourself from malware’ will be treated. Furthermore, two more questions are answered. This questions are ‘How do I detect a malware on my computer?’ and ‘How do I remove a malware?’.

There are a few measures you can take to protect your computer from malware:

- *Install your antimalware software on the computer*. This software can be a very effective defense against malware when it is installed and always up to date. The antimalware software searches for viruses, worms and other malware on the computer. But to be safe and protected against new threats, you should always update the software (Microsoft, 2016).

- *Be careful with emails you open*. If you get an email from an unkown sender or with a strange attachment you should not open it. Emails are often used to infect the computer by open the attachment. To be sure you should not open any attachment you do not expect (Microsoft, 2016).

- *Install a pop up blocker to your internet browser*. Small popup windows can appear in websites which can contain malicious or unsafe code. The blockers can block the appearance of the windows (Microsoft, 2016).

- *Be sure that the operating systems are up to date.* The updates of Windows, Linux or macOS can also help to protect your computer and prevents attacks by malware with closing security holes (Microsoft, 2016).

- *Activate the firewall*. The firewall can help to find suspicious activities on your computer; for example, if something or someone tries to contact or break into the computer. It can also prevent the upload of harmful file or systems (Microsoft, 2016).

- *Delete your Internet cache and your browsing history.* Many browsers have data saved like which websites you visited or information you entered there (e.g. username, birthday or address). You should delete this data from time to time even if they are useful for the user. One case when you should always delete the data is when you are using a public computer and do not want to leave a trace behind (Microsoft, 2016).

4.1 Detecting Malware on the Computer

There are some questions you should answer to decide if there could be a malware on the computer. If the answer is ‘yes’ on one or more questions, then you could be infected (Microsoft, 2016).

The first question is if the computer operates slowly. This a well-kwon indicator of malwares (like in case of the *Morris Worm*). But it is not always the reason of the slow performance. One reason, for example, could be that the computer needs more RAM (memory). Another could be that the hard disk needs defragmenting (Microsoft, 2016).

The next question you should think about is if unexpected events have happened. These events are, for example, message that appear without a reason, the shut down of the computer or programs which are starting without being opened by the user. Some malwares are making programs or the whole computer defective. That are some possibilities that can cause these problems (Microsoft, 2016).

Furthermore, you should check if your computer or external hard disk is working even when it is not supposed to do it (for example when the computer is shut down or in sleeping mode). This could be caused by an email virus which is sending copies of itself by email. To check this, you should have a look at the active light at, for example, the computer or external hard disk or listen to the sound of the system which shows that it is constantly working. This could be a sign of a malware, but it is not in 100% of the cases (Microsoft, 2016).

The last question you should ask yourself is if the antimalware software detected something. That is the most indicating sign of a malware (Microsoft, 2016).

4.2 Removal of Malware

In some cases, a program like Windows Defender removes malware automatically from the computer. In other cases, it has to be removed manually. This can be rather technical. That is one reason why you should try first other programs or ways to get rid of the malware. Another reason is that you need to have good knowledge about the operating system of the computer and handle system and program files of it (Microsoft, 2016).

1. The first step is to run an antimalware software to find the malware by name. If that is not possible (either you do not have a antimalware software or the software does not find the malware) it might still be possible to determine which kind of malware it is by analysing the strange behaviour (Microsoft, 2016).

2. Note down the displayed massages, or if you got the malware via email, note down the subject line or attachment title (Microsoft, 2016).

3. The last step is that you can research the information you gathered from the malware, for example, at an antimalware website for instructions how to remove the malware (Microsoft, 2016).

Once the malware is deleted from the computer, you should reinstall the damaged software or restore a backup if you created some in the past (Microsoft, 2016).

The antimalware software should usually be activated at all time. If that is not the case you should follow the following rules – you should even follow them if it is active (Microsoft, 2016):

- Only treat files whose origin you trust (Microsoft, 2016)

- Do not visit websites that are not secure or trustable (Microsoft, 2016)

- Do not open email attachments from unknown or non-trustable senders (Microsoft, 2016)

5 Conclusion

The objective of this thesis was to analyse the *Morris Worm*. The thesis provides information on the malware in general and on their different types and gives advice on how to deal with them if they infect your computer. It also gives an overview of the current damage caused by cyber-attacks and their future development.

The *Morris Worm* was a malware that appeared on November 2nd 1988 and caused around 98-million-US-dollar damage. The Worm spread through the internet and infected 10% of the computers which were connected to the internet at that time and had the operating system UNIX. The author is Robert Tappan Morris, a student at Cornell University. This type of code has since then been given then name, worm.

Mr. Morris did not mean to harm others with his program. He did not steal any data. But because of a mistake in the function checkother() the computers were overloaded by invisible tasks. The result was that the users could not use their systems probably and it even forced some system administrators to disconnect them from the internet preventing an infection or to shut them down if they were already infected.

The worm’s structure was rather simple (for more details, see Figure 6 and 7). It first tried to infect another host. Next, it reported back to a computer on which computer it was at that moment. Then it tried to crack passwords of the computer. Next, it slept for 30 seconds. Then it entered a while-loop. In the loop the worm first cracked passwords, then it infected other hosts and then it slept 120 seconds. The loop ended only if two conditions were given at the same time or if the worm could not create a new process. The first was that the variable named ‘pleasequit’ has to be one. The other one was that the variable named ‘pleasequit’ has to be bigger than zero.

The mistake in the function checkother() was a small one, but it had impact on the worm and its behavior. In the function checkother() one of the first tasks that the worm was doing was to divide a random number by seven. If the rest was three – thus in one of seven cases – the worm became immortal because this function checked if a copy of the *Morris Worm* is already running on the system. The variable ‘pleasequit’ is added by one which could end the while-loop in the mainloop() function, if a copy is detected. If the worm becomes immortal the modification of the variable pleasequit cannot happen because it exit the function right away.

The immortal percent of the copies did not seem much but they became more and more over the time. They increasingly took processor time which led to an overloading and not responding state of the system. This resembles the effect of a DDoS attack.

If a malware with an effect like this would infect the computer today, it would be much worse - not only because we have more computers now but also because we depend much more on the computer and the internet. It is a disaster, if a company would be without an internet connection even for one day. Nearly every employer could not work because the most work is done with computers or other electronic systems which is connected to the internet. It would be even worse if the malware would infect an airport system. No plane could land or depart at the airport. Hence, the planes would have to fly to another airport or wait until the problem is solved. That can take hours or in case of the *Morris Worm* days. The plane would have no fuel left and crash.

The only good aspect about the *Morris Worm* is that it did not steal any data - not even the passwords that it cracked - unlike the hacking of the company Facebook in 2018. There were almost 50 million Facebook user accounts hacked and data stolen. Today, I would assume, that many hackers steal data because they want to harm others, unlike Mr Morris did.

In order to defend yourself against cyber attacks, there are a few rules you should follow which have a large impact on your security. The first one is that you should always have your antimalware software active and up to date. The second one is that you should only open files whose origin you trust. Another step to prevent getting hacked is to only visit secure and trustable websites. If you get an email but you do not know or trust the sender, never open the attachment. That is the last rule.

But you cannot never be 100% secure. There will always be improvements – on the criminal but also on the protector side. The criminals may find a new way to hack into a system which the programmers of the antimalware software or others did not think about. In addition, they may program a malware which does not have the same effect on the system as others before, for example, an unexpected slow machine or unexpected event happen which your machine.

References

Alvarez Technology Group Inc. (n.d.). *2018 Top Cybercrime Facts and Why You Should Care*. Retrieved from https://www.alvareztg.com/2018-cybercrime-statistics-reference-material/

Boettger, L. (2000, 12 24). *The Morris Worm: How it Affected Computer Security and Lessons Learned by it.* (G. I. Certification, Ed.) Retrieved from https://www.giac.org/paper/gsec/405/morris-worm-affected-computer-security-lessons-learned/100954

Cisco. (2018, June 14). *What Is the Difference: Viruses, Worms, Trojans, and Bots?* Retrieved from https://www.cisco.com/c/en/us/about/security-center/virus-differences.html#1

Frenkel, M. I. (Sept. 29, 2018). Facebook's Woes Rise as Hackers Expose Data of 50 Million Users. *The New York Times*, 1. Retrieved from https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html

Helfrich, J. N. (2018). Security for Software Engineers. In J. N. Helfrich, *Security for Software Engineers* (pp. 55-59). CRC Press.

Indiana University Bloomington. (n.d.). *Indiana University Bloomington*. Retrieved 04 15, 2019, from Things You'll Hear About: https://cs.indiana.edu/docproject/zen/zen-1.0_10.html

Kaspersky Lab. (2003, 11 3). *Morris Worm Turns 25*. Retrieved from https://www.kaspersky.com/blog/morris-worm-turns-25/3065/

Kaspersky Lab IT Encyclopedia. (n.d. -a). *1960s*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/years-1960s/

Kaspersky Lab IT Encyclopedia. (n.d. -b). *1970s*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/years-1970s/

Kaspersky Lab IT Encyclopedia. (n.d. -c). *1980s*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/years-1980s/

Kaspersky Lab IT Encyclopedia. (n.d. -d). *1988.* Retrieved from https://encyclopedia.kaspersky.com/knowledge/year-1988/

Kaspersky Lab IT Encyclopedia. (n.d. -e). *1989.* Retrieved from https://encyclopedia.kaspersky.com/knowledge/year-1989/

Kaspersky Lab IT Encyclopedia. (n.d. -f). *1990*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/year-1990/

Kaspersky Lab IT Encyclopedia. (n.d. -g). *1995*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/year-1995/

Kaspersky Lab IT Encyclopedia. (n.d. -h). *2004*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/year-2004/

Kaspersky Lab IT Encyclopedia. (n.d.). *History of malicious programs*. Retrieved from https://encyclopedia.kaspersky.com/knowledge/history-of-malicious-programs/

Martini, A. (2014, September 8). *The original Morris Worm source code .* Retrieved from GitHub: https://github.com/arialdomartini/morris-worm

Microsoft. (2016, 9 1). *Protect my PC from viruses*. Retrieved from https://support.microsoft.com/en-us/help/17228/windows-protect-my-pc-from-viruses

Nazario, J. (2004). Defense and Detection Strategies against Internet Worms. In J. Nazario, *Defense and Detection Strategies against Internet Worms* (pp. 11-12, 37-68). Boston • London: ARTECH HOUSE, INC.

Oracle Corporation and/or its affiliates. (2010). *The .rhosts File*. Retrieved from https://docs.oracle.com/cd/E19455-01/805-7229/remotehowtoaccess-3/index.html

Oracle Corporation and/or its affiliates. (2010). *The /etc/hosts.equiv File*. Retrieved from https://docs.oracle.com/cd/E19455-01/805-7229/remotehowtoaccess-36082/index.html

Roger, G. (2018, 07 24). *8 types of malware and how to recognize them*. Retrieved from https://www.csoonline.com/article/2615925/security-your-quick-guide-to-malware-types.html

Skoudis, E., & Zeltser, L. (2003). *Malware: Fighting Malicious Code.* Prentice Hall PTR. Retrieved from http://ftp.icm.edu.pl/packages/Hacked%20Team/FileServer/FileServer/OLD%20Fileserver/books/SICUREZZA/Prentice%20Hall%20-%20Malware,%20Fighting%20Malicious%20Code%20(2003).pdf

The Independent IT-Security Institute. (2018, 7 20). *AV Test*. Retrieved from https://www.av-test.org/en/news/the-av-test-security-report-20172018-the-latest-analysis-of-the-it-threat-scenario/