Marko Helenius

A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities



VIRUS RESEARCH UNIT DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES UNIVERSITY OF TAMPERE

A-2002-7

TAMPERE 2002

Marko Helenius

A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities

ACADEMIC DISSERTATION

To be presented, with the permission of the Faculty of Information Sciences of the University of Tampere, for public discussion in the Paavo Koli Auditorium on June 7th, 2002, at 12 noon.

> VIRUS RESEARCH UNIT DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES UNIVERSITY OF TAMPERE

> > A-2002-7

TAMPERE 2002

Supervisor:	Professor Pertti Järvinen Department of Computer and Information Sciences University of Tampere
Opponent:	Professor Klaus Brunnstein Virus Test Center Computer Science Department University of Hamburg
Reviewers:	Professor Juha Kortelainen Department of Computer Sciences University of Oulu
	Assistant professor Cestmir Halbich Department of Information Technologies Czech University of Agriculture Prague

Department of Computer and Information Sciences FIN-33014 UNIVERSITY OF TAMPERE Finland

Electronic dissertation Acta Electronica Universitatis Tamperensis 190 ISBN 951-44-5394-8 ISSN 1456-954X http://acta.uta.fi

ISBN 951-44-5370-0 ISSN 1457-2060

Tampereen yliopistopaino Oy Tampere 2002

Abstract

Computer viruses have become a threat to computer users and computer antivirus products have been developed to facilitate the prevention of computer viruses. Unfortunately, computer antivirus products are not perfect solutions and therefore antivirus product evaluation is needed. One important aspect of computer antivirus product evaluation is analysis of products' virus detection and prevention capabilities. First an introduction to computer viruses and antivirus products' virus detection analysis is presented. We will conclude that analysis of computer antivirus products' virus detection capabilities is a difficult task because of the large number of computer viruses, complex tasks involved with test bed preparation and multiple operations of antivirus products. The author shows that many tasks supporting the analysis of computer antivirus product's virus detection capabilities can be made computer-supported. The author presents a development of computersupported processes, which have facilitated evaluation of antivirus products' virus detection capabilities in various operating environments. These include such processes as automatic virus replication in a controlled environment, automatic evaluation of antivirus programs working actively in the background and automatic processes developed for Windows environment. The major part of the dissertation is devoted to the development phases and self-assessment of a system that can be used for automating these subtasks. Since we consider time saving of the processes as the most critical characteristic, the selfassessment concentrates on efficiency of the processes compared to manually accomplished operations. Problems with different tasks are addressed and also solutions for the problems are provided. The computer-supported processes discussed are especially useful for those who are interested in antivirus product evaluation or virus related aspects of antivirus product quality control. The author shows that the developed processes can save an enormous amount of work and can improve the quality of the evaluation results.

Acknowledgements

Completing this dissertation has required several years of profound research work. This has included, in addition to writing this dissertation, such research work as learning computer antivirus research, conducting computer antivirus product evaluations and developing software and hardware tools for computer antivirus research. I would like to take the opportunity to thank all the people who have helped me during the research and writing process.

I have developed most parts of the antivirus analysis system described in this dissertation, but some ideas for the system as well as some programs were initialised with help of some other professionals. Furthermore, the hardware components were developed externally. I would especially like to thank Kari Laine for part of initial ideas for automated file virus replication, Mikael Albrecht for development of part of software tools and Jarmo Puntanen for implementation of hardware customisations. These people have provided valuable input for the development of the system and thus for this dissertation.

I would like to express my gratitude also to my supervisor Pertti Järvinen, who has continually and encouragingly supported both this dissertation and my work at the Department of Computer and Information Sciences. Furthermore, I would like to thank Tampere Graduate School in Information Science and Engineering (TISE) for a researcher's position, which allowed me to concentrate on this dissertation. I would also like to thank the staff of department of Computer and Information Sciences for support and technical assistance. I would especially like to thank Teppo Kuusisto for configuring a Linux server for the Virus Research Unit's isolated network.

Klaus Brunnstein seems to be the only academic person who has such long experience in supervising antivirus product analyses in an academic environment. His reputation is recognised among antivirus researchers and academics. Therefore I am honoured to have Klaus Brunnstein as my opponent. I would also like to thank my reviewers, Cestmir Halbich and Juha Kortelainen for their professional reviews and useful comments.

In addition, I would like to thank all the computer antivirus researchers with whom I have co-operated. They have provided valuable input for my research work. There are so many researchers that I am not able list them here without forgetting someone. Suffice it to say that if you know some famous antivirus researcher, he or she has probably helped me one way or other and deserves my gratitude. Finally, but not least, I would like to thank my dearest wife for her support in the research work and encouragement to me in writing this dissertation.

1.	Introduction	1
	1.1. Description of topic and its importance	1
	1.2. Definition of the problem investigated	1
	1.3. A short review of previous studies	2
	1.4. Presentation of own approach and its advantages	2
	1.5. Results	3
	1.6. Ethical principles for computer antivirus research	4
	1.7. Structure of this thesis	6
2.		
2.	Method of this study 2.1. Theoretical framework for antivirus product virus detection analysis	' 7
	2.2. Construction of computer-supported processes	
	2.3. Assessment of the system	′ 10
-		
3.	Terminology associated with computer viruses and malicious program code	
	3.1. Classification of harmful program code	11
	3.2. Some special program code classes encountered in virus collections	
	3.3. Classification of computer viruses	14
	3.4. Viruses categorised by their characteristics	
4.	Theoretical framework for antivirus product virus detection analysis	18
	4.1. Theoretical classification of antivirus program's operations	18
	4.1.1. Classification of antivirus programs	19
	4.1.2. Correct and false virus detection by antivirus programs	19
	4.2. Antivirus product virus detection analysis	21
	4.2.1. Virus scanners	22
	4.2.1.1. Known virus scanners	22
	4.2.1.2. Heuristic scanners	23
	4.2.1.3. Memory resident known virus scanners	
	4.2.1.4. Memory resident heuristic scanning	25
	4.2.2. Integrity checkers	
	4.2.3. Behaviour blockers	
	4.2.4. Memory resident integrity checkers	26
	4.3. Antivirus product virus detection analysis methods	
	4.3.1. Detecting known viruses	
	4.3.2. Preventing known viruses	27
	4.3.3. Detecting unknown viruses	27
	4.3.4. Preventing unknown viruses	28
	4.4. Different virus types in the test bed	
	4.4.1. File viruses	
	4.4.2. Boot sector viruses	
	4.4.3. Macro viruses	
	4.4.4. Script viruses	29 20
	4.4.5. Multipartition viruses	
	4.4.6. Polymorphic viruses	29 20
	4.4.7. Companion viruses	
	4.4.8. Stealth viruses	
	4.4.9. Linking viruses 4.4.10. Memory resident viruses	
	4.4.10. Nemory resident viruses 4.4.11. Self-distributing viruses	
	4.4.11. Sen-distributing viruses	
	4.5.1. False alarms	

	4.5.2. The problem of bias	32
	4.5.3. Viruses found in the field versus laboratory viruses	33
	4.5.4. Determining the potential threat posed by each virus	35
	4.5.5. The threat of unknown viruses	36
	4.5.6. The ever growing number of different viruses	36
	4.5.7. Regular access to antivirus products	36
	4.5.8. Reliability problems	
	4.5.9. Replicates of viruses	
	4.5.10. Ensuring that each virus in a test bed is a true working virus	
	4.5.11. Differentiating virus variants	
5.	Development of computer-supported methods for computer antivirus	
	product virus detection analysis	41
	5.1. Development phases of the Automatic and Controlled Virus Code	
	Execution System	41
	5.2. The initial idea	43
	5.2.1. Manual or computer-supported virus replication?	43
	5.2.2. Make or buy?	
	5.2.3. Emulation or hardware implementation?	44
	5.3. First specification	
	5.4. First implementation	46
	5.4.1. Simple implementation	
	5.4.2. Automated cold boot	46
	5.4.3. Improved replication	47
	5.5. Automatic boot sector virus replication	47
	5.5.1. Boot device selection	47
	5.5.1.1. Selecting a boot device from the network card	48
	5.5.1.2. Selecting a boot from a floppy diskette	48
	5.5.1.3. Selecting a boot from 3.5 or 5.25-inch floppy diskette	48
	5.5.2. The implementation	
	5.6. Improved file virus replication	51
	5.7. Automated processes for the MS-DOS environment	
	5.7.1. File virus detection analysis of memory resident scanners	
	5.7.1.1. File copy method	
	5.7.1.2. File execution method	56
	5.7.2. Boot sector virus detection analysis of memory	
	resident scanners	_57
	5.7.3. Automatic virus detection analysis of MS-DOS behaviour blockers	
	5.8. Multipartition virus replication of file and boot sector viruses	
	5.9. Automatic macro virus replication	58
	5.9.1. Solving the user action problem	58
	5.9.2. Implemented replication process	60
	5.9.3. Other replication environments for macro viruses	
	5.10. Automating other tasks in Windows environment	<u>. 62</u>
	5.10.1. Automatic replication of file viruses for Windows	
	5.10.2. Boot sector virus detection analysis in Windows environment	
	5.10.3. Memory resident scanners for Windows environment	
	5.10.4. Automatic replication of self-e-mailing viruses	
	5.11. Other possible tasks	04

5. Self-assessment of the computer-supported processes 65	
6.1. Comparison with other computer-su	pported systems 65
6.2. Comparison with manual virus repli	cation processes 67
6.2.1. Manual file virus replication	68
6.2.1.1. Results from the Virus F	esearch Unit's 80286 computer68
6.2.1.2. Results from the Virus R	esearch Unit's 90 MHz Pentium
computer	69
6.2.2. Manual boot sector virus repli	cation69
	n70
6.2.4. Manual replication of file viru	ses infecting Windows executables 70
6.3. Automatic virus replication processe	28 <u>71</u>
6.3.1. Automatic file virus replicatio	n71
6.3.1.1. Results from the 80286 c	computer 72
6.3.1.2. Results from the Pentium	n computer 73
6.3.2. Automatic boot sector virus re	plication73
6.3.3. Automatic macro virus and W	indows executable virus replication 74
6.4. Comparison of manual and automati	c file virus replication 75
7. Discussion	78
7.1. On the importance of the results	78 78
7.2. Limitations	
7.3. Recommendations to practitioners	
7.4. Recommendations to researchers	85
References	
Appendix 1	

1. Introduction

1.1 Description of topic and its importance

The research domain of this dissertation can be classified as a subfield of the computer antivirus research domain, whereas the computer antivirus research domain can be classified as a subfield for the computer security domain. Computer security and computer antivirus research have become important research domains as the connectivity, computerisation and complexity of computer systems have increased. Existing computer systems and applications had not been designed as secure as they could have been (see, for example, Stojakovic-Celustka 2000) and this has not only allowed misuse, but also forced subsequent patching of known security deficiencies (see, for example, CERT 2001a and Christopher 1996).

The computer virus problem is a good example of this and as Coursen (1996) has demonstrated, the cost of a virus incident can be high. Coursen proposed the following:

"This one incident (which incidentally was handled very well), involving 325 employees over a four-hour period of time realized a loss in excess of US\$10,000.00. Add to that the value of non-recoverable data, the time required to scan every diskette in the company, and other miscellaneous operational changes, and the cost of the incident comes into focus."

Open and insecure architectures have allowed computer viruses to exist. Where computer viruses exist, software solutions against computer viruses have also been innovated and those innovations need to be assessed in order to provide information about the efficiency of the solutions. This work presents processes that have been designed to facilitate assessing computer antivirus products' virus detection capabilities.

1.2 Definition of the problem investigated

We have three research questions in this dissertation. At first we must know what computer antivirus detection analysis is about (see Appendix 1, which contains definitions of some terms used in this dissertation). Therefore we will present a theoretical framework for computer antivirus product virus detection analysis. We note that manual virus detection analysis is time-consuming and therefore we will examine the second question: Can computer-supported processes be developed for computer antivirus product virus detection analysis? The third question is, if the computer-supported processes can be developed, how effective they are. In this dissertation we try to answer these three questions.

1.3 A short review of previous studies

Although computer antivirus research and computer antivirus product evaluation (see Appendix 1) are new and challenging research domains there exist some important research outcomes, which are worth mentioning. Cohen (1986) was the first to establish formal definitions for computer viruses. Bontchev (1998) published a doctoral thesis on the methodology of computer antivirus research. Brunnstein (1999) presented a classification scheme of malicious software. The Virus Test Center has been publishing antivirus product virus detection analyses (1994-2002). In addition, we at the Virus Research Unit have published antivirus product virus detection analyses (Helenius 1994a, 1995b, 1996b, 1997 and 1999a). Furthermore, several papers concentrating on antivirus product evaluation have been published in EICAR conferences, Virus Bulletin conferences and information security conferences.

The antivirus product analysis processes described in this dissertation have been developed without knowing about other implementations and the processes developed are as such novel innovations. However, there also exist other systems that have been developed parallel to our system. Swimmer presented Virus Intrusion Detection Expert System (1995), Leitold presented Automatic Virus Analyser System (1995), IBM developed an Immune System Concept (Kephart et al. 1997) and Whalley presented a potential system that allows automatic replication of viruses that replicate by using the Internet (2000). Previous studies have mainly described the systems at general level as is understandable, because the implementations of the systems are typically business secrets.

1.4 Presentation of own approach and its advantages

The work presented in this dissertation is largely based on my research work at the Virus Research Unit, which is located at the Department of Computer and Information Sciences at the University of Tampere. My research has concentrated on computer antivirus research. This includes such research work as conducting analyses of computer antivirus products and the development of tools for computer antivirus research.

I consider research in the field of computer antivirus product evaluation important, because there have been only few studies in this field so far which have discussed the subject in detail. Papers discussing computer antivirus product evaluation have seldom tried to provide solutions to the problems they have addressed. One important objective of this dissertation is to advance research in the computer antivirus domain. We will establish a theoretical framework for computer antivirus product virus detection analysis. We will describe the development of computer-supported processes more specifically than previous studies and therefore provide valuable information. Furthermore, the efficiency of the systems has so far not been proved. In the thesis we will prove that this system is efficient and that its development is profitable provided that usage of the processes is continuous.

1.5 Results

The important question is: Can the processes be built? (see March and Smith 1995, p.258 and Järvinen 1999, p.59) In this thesis we will demonstrate how a system implementing the processes was built and thus we will demonstrate that the system has already been built. From this it follows that the system can be built. The subsequent question is: How good is the system? More specifically we may ask: What are its advantages and what are its disadvantages compared to competing processes? As at the moment of writing this thesis there does not seem to exist detailed information about other competing systems, we will briefly discuss other systems and concentrate on comparing our system with manual processes. We will concentrate on the efficiency of the system and show that the system can save multiple times more time and resources in the long run than manual processes applied for the same tasks. Moreover, one important achievement of this thesis is the theoretical framework established for computer antivirus product virus detection analysis.

1.6 Ethical principles for computer antivirus research

Those who are not familiar with computer antivirus research may not know the general ethical rules prevailing among computer antivirus researchers. Therefore the currently¹ prevailing ethical principles are briefly discussed here. Furthermore, computer viruses are usually conceived as harmful and hazardous instruments, if treated carelessly. Therefore the ethical issues of computer antivirus research cannot be ignored and antivirus researchers must apply safety policies and take ethical responsibility in computer antivirus research. After all, ethical responsibility is what distinguishes virus writers and distributors from responsible computer antivirus researchers who fight against viruses.

EICAR (Europian Institute for Computer Antivirus Research) is an organisation whose objective is to "combine universities, industry and media plus technical, security and legal experts from civil and military government and law enforcement as well as privacy protection organisations whose objectives are to unite non-commercial efforts against writing and proliferation of malicious code like computer viruses or Trojan Horses, and, against computer crime, fraud and the misuse of computers or networks, inclusive malicious exploitation of personnel data, based on a code of conduct" (EICAR 1999a). EICAR arrogates that each member must recognise the EICAR's code of conduct (1999b). The code of conduct has the following points:

¹ In this work we define the word current to mean the time of completing this dissertation, which is May 2002.

- Total abstinence from activities or publications, which could cause or foster panic, i.e. no "trading on people's fears".
- Abstaining from the loud and vociferous superlatives and factually untenable statements in advertising, e.g. "all known and unknown viruses will be recognised".
- Information which is suited for the development of viruses as well as other malicious program code will not be published or given to a third party. Exchange of such information with institutions, companies and persons is excepted, which are responsibly researching or are active in combating in this sector.
- The recognition of the EICAR code of conduct is a requirement for membership.

The purpose of the first two sentences is to respond to the exaggerating marketing that antivirus producers may indulge in order to advance antivirus product selling. The object of the third sentence is to prevent enlarging the malware problem.

There are also some ethical issues that are not indicated in the code of ethics and which seem to be recognised by the antivirus community. Writing or creating new viruses is considered improper as well as selling or buying viruses. For example, rewarding customers for providing viruses is considered improper, because it may stimulate creation of new viruses. Furthermore, antivirus researchers should take care that viruses are not given or leaked to outsiders. Making viruses publicly available or transmitting viruses via insecure channels is considered improper. If there is a possibility that the submitted information could leak, encryption is required. Insecure channels include, for example, transmitting viruses via the Internet or via postal mail.

It is important to note that the principles do not concern just executable viruses, but also virus source codes and instructions or tools for virus creation. Furthermore, the same principles also concern other malicious software.

Because most of the responsible computer antivirus researchers are members of the EICAR, or at least, recognise EICAR's significance, EICAR's code of conduct outlines general principles of the computer antivirus research ethics. It can be argued that the code of ethics cannot be always precisely interpreted, but it shows the general direction for computer antivirus research ethics. As well as other computer antivirus researchers, those evaluating antivirus products should adopt a seriously ethical view of the research.

One recent proposal for a code of conduct worth mentioning is the AVIEN's (Anti-Virus Information Exchange Network) Anti-Virus Professionals Code of Conduct (AVIEN 2001). The content of the code of conduct is the following:

(i) DO NO HARM

I will not write and deliberately release any code with malicious intent. With malicious code being defined as not only code that does direct or indirect damage to systems and data, but also code that has undesirable secondary consequences such as risk of embarrassment to or punishment of the victim.

I will not write replicative or destructive code unless I am convinced that it is necessary for internal research or testing purposes as required and defined by my professional activities. If I regard it as necessary to write such code, I will do so under secure and strictly controlled conditions, and I will not publish such code. Nor will I share it unless it is absolutely necessary, and then only with individuals whose competence and adherence to this code of conduct or an equivalent is beyond question. I will not keep copies of such code for any longer than is strictly necessary, and only under secure and strictly controlled conditions.

I will not deliberately damage live data. Nor will I alter any data except as authorized by the owner of those data.

I acknowledge that the public release of Malware, even for benevolent purposes such as advising potential victims of vulnerabilities in their systems, is never beneficial if it involves unauthorized access or modification to systems, even if the quality and safety in use of the code could be guaranteed under all circumstances.

(ii) DUTY OF CONFIDENCE

I will treat as confidential all data entrusted to my care. I will not divulge my client or employer's identification, or claim to act as their representative, except with their expressed consent, or where an overriding legal or moral obligation exists.

(iii) DUTY TO BEHAVE RESPONSIBLY

I will behave at all times in accordance with all applicable laws, policies, and codes of conduct required by AVIEN and any other organization with which I am affiliated.

Other than for publicly accepted legitimate development or research as part of my professional activities in understanding and/or creating defenses against malware, I will not intentionally trade, solicit, or transmit malware, or encourage these activities. I will always discourage such activities other than for publicly acceptable legitimate development, testing or research. I will not pass on malicious code to anyone whose competence and integrity is in doubt.

(iv) DUTY OF CARE

Malware entrusted to me in my professional capacity will be handled with the utmost care and respect for their capabilities for harm, in order to prevent infection or dissemination.

I will assume responsibility for viral incidents when charged with their management, irrespective of whether they result from any action of mine.

If contacted with details of a possible infection, I will proceed as if there is a definite, proven infection until it can be proved otherwise. If any system in my charge is infected, I will advise all individuals or organizations who may have been a source of infection, or who may have received malicious code as a result of contact with those systems.

(v) DUTY TO INFORM AND EDUCATE

I will dispel Malware hype, myths and misinformation through education. I will not claim knowledge or ability beyond my actual capabilities. I will not use Malware-related hype or fear-mongering to promote any company, any product, or myself.

I acknowledge and recognize that Virus eXchange (vX) web sites and bulletin boards only further the malware problem. I will not validate their existence by frequenting them, other than for ethically acceptable research into their activities. When asked, I will support and assist authorities in discouraging and suppressing vX activity wherever possible.

I understand and agree to this Code of Conduct and pledge to act in an ethical and professional manner, as outlined above.

One important observation is that the code of conduct takes a stand on writing malicious code and on making viruses available. Since the code of conduct is a recent outcome, we do not know how well it will be adapted. Bechtel (2001) has discussed the background of the code of conduct.

1.7 Structure of this thesis

In Chapter 2 we will discuss the method of this study. In order to provide a theoretical background we will continue by establishing definitions and restrictions concerning computer viruses and computer antivirus product evaluation. In Chapter 3 we will discuss the terminology associated with malicious program code and computer viruses. In Chapter 4 we will establish a theoretical framework for antivirus product virus detection analysis. We will discuss theory of antivirus product's operations, how antivirus products' virus detection should be carried through, how a virus test bed (see Appendix 1) should be constructed and we will discuss some problems associated with antivirus product evaluation. In Chapter 5 we will discuss the development phases of the developed computer-supported processes for computer antivirus product's virus detection analysis. In Chapter 6 we will compare the efficiency of computer-supported processes with manual processes. Finally, in Chapter 7 we will draw conclusions and discuss the limitations of this dissertation and suggest possible future steps.

2. Method of this study

In this work we will present a system that is capable of automating several tasks that can be used for computer antivirus product virus detection analysis. However, in order to provide a theoretical background we will at first present definitions and restrictions concerning computer viruses and antivirus product evaluation.

In constructive research when we are building an artefact the important question is: Can the artefact be built (see March and Smith 1995, p.258 and Järvinen 1999, p.59)? Therefore in this dissertation we will present the development process of the system as well as other computer-supported virus detection analysis processes used in the Virus Research Unit's antivirus scanner analyses (see Helenius 1994a, 1995b, 1996b, 1997 and 1999a). Thus we will prove by demonstration that computer-supported processes can be built. The subsequent question is: How good is the system? Therefore assessments of the computer-supported processes are conducted. Although there also exist other characteristics, we will concentrate on efficiency, because we see this as the most critical characteristic.

2.1 Theoretical framework for antivirus product virus detection analysis

A theoretical framework is needed to understand computer antivirus product virus detection analysis. The framework is mainly constructed from experiential knowledge gathered during my research work at the Virus Research Unit. Some theories have been developed from previous papers and studies. For example, I discussed some problems and solutions associated with antivirus product evaluation in a conference paper at the EICAR conference 1996 (Helenius 1996). From the theoretical framework for computer antivirus detection analysis we can move to the construction of computer-supported processes.

2.2 Construction of computer-supported processes

At first we should note that some development phases of the processes were presented in conference papers (Helenius 1995a, 1998a and 1998b). However, my choice was to rewrite the development phases of the processes in a more detailed and consistent development description.

The computer-supported processes were constructed by starting from a simple implementation and gradually extending to more complex implementations. When one stage was discovered to be viable, more features and new processes were constructed. From automatic virus replication processes a system named as Automatic and Controlled Virus Code Execution System was constructed. The system was first presented at the EICAR conference in 1995 (Helenius 1995a). At first the system was used for automatic virus replication, but it also enabled the construction of other processes.

From the beginning the following general principles were established for the system because of safety and flexibility requirements:

- 1) The system must be isolated in such a way that a possibly escaped virus cannot cause harm to external computer systems.
- 2) The system must be designed as much as possible in such a way that a malicious code executed in a controlled environment cannot harm the system.
- 3) The system should be designed to be flexible in order to allow flexible future development.
- 4) The system should be designed to work as continuously as possible

In order to meet the first condition the system was isolated from external connections and also integrity of executable files of the system was checked. In order to meet the second condition I carefully prepared for possible vulnerabilities of the system. In order to fulfil the third requirement the software components of the system were designed as flexible as possible. In order to meet the fourth condition such problems were solved that would have precluded the system from working continuously.

The Automatic and Controlled Virus Code Execution System development started from an initial idea for automatic file virus replication. The implementation idea is presented in Figure 1. At the same time this was the target state. The target state was achieved, but other tasks were also perceived possible with the help of the system and this generated new ideas and therefore new target states. Furthermore, while the number of viruses infecting new types of object became large the need for automatic processes for these viruses also arose. When macro viruses appeared automatic macro virus replication and automatic processes for evaluation of macro virus detection were constructed. The system's capabilities to replicate macro viruses in a controlled environment was presented at the EICAR conference in 1998 (Helenius 1998a). Similarly when self-e-mailing viruses constituted a threat the need for automatic replication and evaluation arose. The system development resembled Floyd's project model of STEPS (Software Technology for Evolutionary Participative System Development, Floyd et al. 1989, p.57). Usage of the system sometimes generated unpredictable situations and thus it gave feedback that was used for improving the system. While the system was used for antivirus product analysis, the improvements were implemented step by step until the system was stable. Sometimes there were programming errors, which caused problems and which had to be fixed.

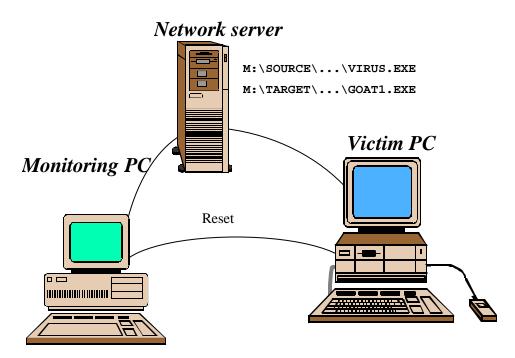


Figure 1: Implementation idea of the automatic and controlled virus code system

A typical programming error was a timing error. For example, if a virus active in a computer impaired the computer's performance, this had to be observed during the system development. The infected computer needed to have enough time to execute all required operations. A tragic example of a programming error is a situation where a computer was logged into a network with write rights to the wrong network directories after the virus code had been executed.

Sometimes there were unforeseen situations or required part of the implementation was missing. For example, certain malicious software could change CMOS memory's (see Appendix 1) content. This had to be fixed by automatically restoring the original content of the CMOS memory. Another example is a malicious program that destroyed the contents of the hard disk in such a way that it had to be low level formatted. This was fixed by automatically low level formatting the hard disk, if normal recovery did not recover the original system.

2.3 Assessment of the system

Since we do not have accurate technical details of other competing systems or processes we decided to briefly compare the efficiency of the processes developed with manual processes required for the same tasks. Therefore I conducted controlled laboratory experiments measuring the efficiency of manual processes.

We define the difference between manual and automated processes such that manual processes do not have customised hardware implementations to automate all required operations whereas automated processes do not need human assistance once initiated. The argument for assessing the manual processes by myself is the expertise achieved from the research area and the capability to construct test conditions as correctly as possible.

The manual processes were executed in such a way that no customised hardware parts were utilised. However, such semi-automatic tasks were included in the manual replication which did not require hardware customisation and which were likely to be used in manual replication environments. This includes using batch files for executing goat files (see Appendix 1), automatic recovery of the fixed disk, checksum calculation, obtaining the sample file from the network server and saving changed objects to the network server. The intention was to estimate the maximum human processing efforts and therefore the intention was not to measure the human weariness that a monotonous work can cause. Therefore the processes were short enough to exclude weariness.

The same computers that were used with manual processes were used with automatic processes. The argument for using the same computers was to eliminate the effects that different computers would have had on replication time. The sample files were obtained from a virus collection containing possible viruses. In other words the sample files of the virus collection were not proved to contain viruses. To summarise, in the assessment section of this thesis, the controlled laboratory experiment setting was imitated as closely as possible.

We will present the results of experimental manual virus replication processes, the results from automatic virus replication processes and then compare the results. The replication speed of manual processes was recorded by using a program that recorded the process starting time and the sample file name for each replication process. The replication speed of automatic processes could be gathered from log files created during usage of the system.

3. Terminology associated with computer viruses and malicious program code

Before one can understand the preconditions for professional computer antivirus product evaluation, one must be familiar with certain terms associated with malicious program code and computer antivirus products. Furthermore, terms discussed in this chapter will be continually referred in this dissertation. Therefore it is important to internalise the definitions of these terms. We will at first present a classification of harmful program code, then we will present some program code classes that are typically encountered in virus collections. Finally, we will classify computer viruses first based on infected object and then based on virus characteristics.

3.1 Classification of harmful program code

I discussed the problematic nature of defining malicious software in a conference paper published at the EICAR conference in 1999 (Helenius 1999). According to conversations with participants at several conferences and educational meetings, students and journalists, it seems that even people familiar with computing often have an unclear and even controversial understanding of the terms associated with malicious program code. In fact, an exact definition for the term malicious program code or malicious software (=malware) has not been agreed on even among computer antivirus researchers. Before we can refer to specified terms we must define the main terms associated with malicious program code and computer virus prevention.

I would like to emphasise that for some of the following terms there is no common agreement on exact definition, but the main functions for each type of code and software have been generally recognised by computer antivirus researchers. One reason for the difficulties is that it is impossible to say in all given circumstances whether a given program is malicious or not. For example, a program that formats hard disks can be considered either as harmful or useful, depending on the purpose for which the program is used. According to discussions in VForum (a virus specific e-mail discussion list dedicated for computer antivirus researchers) the following definitions seem to meet some agreement.

We will use the term "program code" rather than "program" to emphasise that the definitions may concern partial programs, too. The lists of program code classes associated with some of the definitions may not be exclusive and thus there may also exist other program code classes that belong to the definition. Furthermore, sometimes there may exist so-called grey areas where it is difficult to define whether a program code belongs to a category or not. Figure 2 illustrates different program code classes associated with harmful program code.

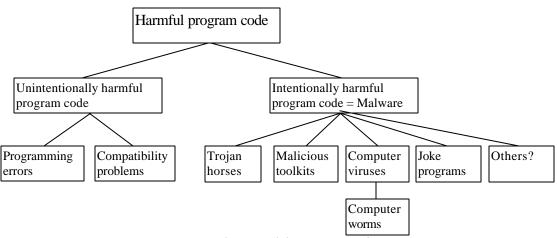


Figure 2: Harmful program code

Harmful program code (constructed from Brunnstein's definition, see Brunnstein 1999): Refers to any part of a program code which adds any sort of functionality against the specification or intention of the system. We define the system to include all installed hardware and software. Harmful program code includes all program parts which are against the system's specification or intention. Harmful program code can be divided into unintentionally harmful program code and intentionally harmful program code. The latter is synonymous with malicious program code.

Unintentionally harmful program code: Refers to a program code which has been inadvertently made harmful. This includes such a program code that is harmful because of programming errors or compatibility problems.

Intentionally harmful program code = Malicious program code: Refers to a program code which has been deliberately made harmful. This includes such program code classes as Trojan horses, computer viruses, joke programs and malicious toolkits. The list may not be exclusive.

Computer virus: Refers to a program code which has a capability to replicate recursively by itself. Computer viruses may include operations, which are typical for Trojan horses and malicious toolkits, but this does not make such viruses Trojan horses or malicious toolkits.

Computer worm: Refers to an independent program code which has a capability to replicate recursively by itself. Independent means that a computer worm does not have a host program which the worm has infected or replaced by its own code. Computer worms are a subgroup of computer viruses. Computer worms may include operations which are typical for Trojan horses and malicious toolkits, but this does not make such worms Trojan horses or malicious toolkits.

Trojan horse: Refers to a self-standing program code which performs or aims to perform something useful, while at the same time intentionally performs, unknowingly to the user, some kind of destructive function (constructed from Bontchev's (1998, p.14) definition). Self-standing means that, in distinction to viruses, the program code does not have the capability to replicate by itself. The program code may be attached to any part of a system's program code. Trojan horses may include operations which are typical for malicious toolkits but this does not make such Trojan horses malicious toolkits.

Malicious toolkit: Refers to a toolkit program which has been designed to help such malicious intentions, which are aimed against computer systems. This includes such programs as virus creation toolkits and programs, which have been designed to help hacking.

Joke program: Refers to a program which imitates harmful operation, but does not actually accomplish the object of imitation and does not contain any other malicious operation.

3.2 Some special program code classes encountered in virus collections

The so-called virus collections often contain other program code classes than viruses. By a virus collection we mean a set of suspicious files that are not verified to contain viruses. Typically virus collections contain, in addition to already defined program code classes, special types of program code, which we will define next. The list may not be exhaustive.

Dropper: Refers to a Trojan horse or a malicious toolkit, which installs a virus in some part of a system.

First generation virus: Refers to the first replication generation of a virus. Often a virus is distributed as a known sample containing a first generation virus and sometimes later replicates of a virus are different from the first generation.

Intended virus: Refers to a program code, which has been designed to work like a virus, but for some reason the program code is not able to replicate recursively and thus intended viruses do not belong to the virus category. Intended viruses are often encountered in poorly organised virus collections.

Innocent program: Refers to a program, which has no malicious specification or intention. The definition of an innocent program is included here, because innocent programs are often encountered in poorly organised virus collections.

3.3 Classification of computer viruses

Computer viruses can be classified into four basic classes by infected objects. A virus can also infect several different types of objects. Figure 3 illustrates these virus categories.

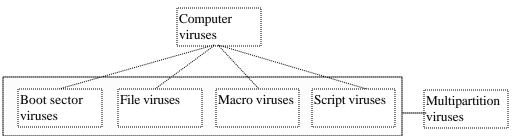


Figure 3: Computer viruses categorised by infected object

File virus: Refers to a virus, which replicates on executable files.

Boot sector virus: Refers to a virus which replicates on boot sectors of floppy diskettes and/or on boot and/or partition sectors of hard disks.

Macro virus: Refers to a virus which uses application macros for replication.

Script virus: Refers to a virus that uses operating system scripting language for replication. This includes such as DOS batch file viruses, Visual Basic Scripting (VBS) language viruses (For more details see, for example, Zwienenberg 2001) and Unix shell script viruses.

Multipartition virus: Refers to a virus which utilises at least two of the previous replication methods. For example, some viruses can replicate on both executable programs and boot sectors or some viruses can infect both executable files and document files.

3.4 Viruses categorised by their characteristics

Each of the previously defined virus types may include the following characteristics. The same virus may contain several characteristics. For example, a file virus can be classified as memory resident virus, stealth virus, self-distributing virus, e-mailing virus and polymorphic virus. However, a virus must always be a direct action virus or a memory resident virus. These two categories may also apply jointly. Figure 4 illustrates the categories.

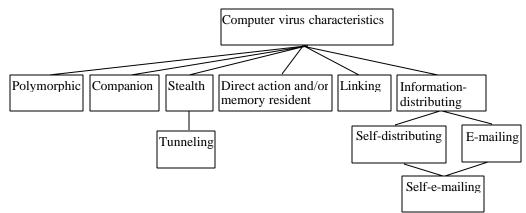


Figure 4: Viruses categorised by their characteristics. A virus may combine these characteristics, but a virus must always be a direct action virus or a memory resident virus.

The list of categories may not be exhaustive, because different practices can be used to classify viruses. Furthermore, there may appear so far unknown characteristics that current² viruses are not using. We will next present the definitions for the different categories in Figure 4.

Memory resident virus: Refers to a virus which remains active on a computer's central memory after the virus code has been executed.

Direct action virus: Refers to a virus which does not remain active on a computer's central memory after the virus code has been executed. Replication occurs during execution of the virus code. A virus may be both a direct action virus and a memory resident virus, if the virus uses both of these methods. For example, a virus may casually install itself in central memory or it may remain in memory, but later uninstall itself from memory.

Information-distributing virus: Refers to a virus which has an intentionally built capability to distribute information from a local system to a remote system via some solid information channel. A remote system refers to a computer system other than the system in which the virus is executed. The information distributed does not need to be, but may be meaningful.

² In this work we define the word current to mean the time of completing this dissertation, which is May 2002.

Self-distributing virus: Refers to a virus that has an intentionally built capability to distribute itself from a local system to a remote system via some solid information channel. Self-distributing viruses are a subgroup of information-distributing viruses. For more information about self-distributing viruses see Tocheva (2001).

E-mailing virus: Refers to a virus that is able to send e-mail. The e-mail sent can be the virus itself, but does not need to be. E-mailing viruses are a subgroup of information-distributing viruses.

Self-e-mailing virus: Refers to a virus that has a capability to send itself via email. Self-e-mailing viruses are a subgroup of e-mailing, self-distributing and information-distributing viruses. Self-e-mailing viruses are often called as mass-mailing viruses, but in this dissertation these viruses are called e-mailing viruses in order to indicate that the number of e-mails distributed does not need to be massive.

Companion virus: Refers to a virus which exploits system specific file execution order for replication. For example, in MS-DOS executable files with the extension COM are executed before executable files with the extension EXE, if the files have the same name, but different extension and they reside in the same directory. Furthermore, in many operating systems the search path defines the order in which executable code is searched and executed. A virus can utilise this file execution order in such a way that the virus code is executed before the actual program code. A companion virus can also rename or store the infected object in some other location or use alias type shortcuts.

Polymorphic virus: Refers to a virus which has a variable appearance in different replication generations of the virus. Typically polymorphism is achieved by using variable encryption, variable instruction order, variable instructions, do-nothing instructions or a combination of these methods. (For a more detailed description and classification of polymorphic viruses see Bontchev 1998, pp. 65-74).

Stealth virus: Refers to a virus which hides at least part of the changes it has made in the system. (For a more detailed description and classification of stealth viruses see Bontchev 1998, pp. 45-54). Typically stealth viruses are memory resident viruses in order to be able to efficiently hide changes in a computer system. However, a stealth virus does not need to be memory resident in order to hide certain changes. For example, macro viruses can hide menu selection that enables viewing macros without being memory resident.

Tunneling virus: Refers to a virus which has been designed to prevent programs executed after the virus code from detecting changes in a system. This method can be used, for example, to bypass such antivirus programs that are executed after a virus. The bypassing is based on the assumption that the virus code has been executed before the antivirus program and thus the virus has access to the system in such a way that it can prevent other programs from working as they are supposed to. (For a more details on tunneling viruses see Bontchev 1998, pp. 56-62) As tunneling viruses try to hide the changes they have made in the system, tunneling viruses are classified as a subclass of stealth viruses.

Linking virus: Refers to a virus which utilises program code linkage for replication. For example, in MS-DOS each directory contains directory entries. The directory entries contain such information as file attributes and files' physical addresses on the disk. A virus can change the address to point to itself and return the control back to the original program after execution of the virus code.

4. Theoretical framework for antivirus product virus detection analysis

As long as there have been antivirus products, there have also been evaluations of antivirus products. Antivirus product evaluation (see Appendix 1, which contains definitions of some terms) differs from typical software evaluation (Virus Test Center 2001). End users and typical magazine evaluators are not able to evaluate accurately the most critical part of antivirus products. They do not have the knowledge and resources to estimate how well antivirus products can prevent or detect viruses. Therefore professional antivirus product virus detection analysis is needed.

One requirement for analysing antivirus products is that there should be a well maintained virus test bed (see Appendix 1). An antivirus analyser should also know exactly what the collection contains. In addition, an antivirus analyser should know how antivirus products work and what their vulnerabilities are. In other words, one must know how antivirus products can prevent viruses and how viruses can attack antivirus products (For a detailed description of different methods viruses can use for attacking antivirus programs, see Bontchev 1998, pp. 192-221). All this makes antivirus product evaluation very demanding.

In the following sections I will describe my own experiences of analysing computer antivirus products from a technical point of view. I have concentrated on examining what the problems with analysing computer antivirus software are and how some of these problems could be solved. Disinfection techniques are not taken up for discussion in this thesis as disinfection is its own problem domain. Therefore I have decided to exclude this part of antivirus programs from the domain of the dissertation.

4.1 Theoretical classification of antivirus program's operations

Our next step is to discuss the theoretical classification of antivirus products operations. We will present a general classification of antivirus programs. Then we will examine how antivirus programs can fail. In the next section we will use the findings of this section when discussing theoretical classification of antivirus product virus detection analysis.

4.1.1 Classification of antivirus programs

Computer antivirus programs can be classified by their behaviour (Helenius 1994c, pp. 25-26). The definition has been extended from Kauranen's (1990, pp. 25) definition. Antivirus programs are often designed to identify a virus, in which case the program detects a virus known to the program. Moreover, a program may be designed to find a virus based on the general behaviour of viruses. In this latter case the virus is not known to the program can give some information based on the behaviour of the virus. Another aspect is that a product can detect a virus after infection has occurred or before the infection to new objects occurs. From the identification and prevention mechanisms we can construct a two-dimensional table (Table 1). However, it is important to note that antivirus products typically contain several types of different program and the programs are often integrated.

Identifying and non-preventing antivirus programs like known virus scanners	Identifying and preventing antivirus programs like memory resident known virus scanners
Non-identifying and non-preventing antivirus programs like checksum calculation programs and heuristic scanners	Non-identifying and preventing antivirus programs like behaviour blockers, memory resident checksum calculation programs and memory resident heuristic scanners

Table 1: Two-dimensional classification of antivirus programs

4.1.2 Correct and false virus detection by antivirus programs

An antivirus program may fail in its virus detection both by false positives and false negatives. Table 2 demonstrates the possible correct and incorrect operations of antivirus products when virus detection is attempted.

	Virus exists on the inspected object	No virus exists on the inspected object
Correct operation	Alert \rightarrow Correct positive	No alert \rightarrow Correct negative
Incorrect operation	No alert \rightarrow False negative	Alert \rightarrow False positive

Table 2: Correct and incorrect operations of antivirus product's virus detection

On the one hand an antivirus product can correctly find a virus (correct positive) or correctly perceive that there is no virus on the inspected object (correct negative). On the other hand an antivirus product may fail to find a virus (false negative) or notify about a virus when in reality there is no virus on the inspected object (false positive).

False positives are often called false alarms (see Appendix 1). Whenever a virus has been assumed to be found a user or an administrator is alerted and the probable assumption is that the inspected system area has been infected. In the opposite case there are no user alerts and a user probably assumes that the inspected system area is clean from viruses. In both cases the assumption can be incorrect and a user has received incorrect information because the antivirus product has failed in its virus detection.

Virus detection analysis concentrates on virus detection and false negatives, but a thorough antivirus product evaluation should also estimate sensitivity to false positives especially, if default scanning modes are changed, because increased sensitivity to find viruses also increases the possibility of false positives. As an extreme example, if there is no correct negative verification, a product that would report every inspected object as infected, would get a full score.

In addition to the cases presented in the Table 2 there is also the possibility that an antivirus product does not try to find a virus from the object (for example, file or boot sector) that is infected although the object belongs to the inspected area. Table 3 demonstrates the correct and incorrect operations of an antivirus product in this case.

	Uninspected objects on the inspected area
Correct operation	No known infection possibility
Incorrect operation	Known infection possibility

Table 3: Correct and incorrect operations of antivirus product's virus detection when objects on the inspected area are not inspected

In Table 3 known infection possibility means that an infection possibility is known to the product, but the product for some reason does not try to inspect the possibly infected objects in a selected system area. In other words, the product would be able to find possible infections and it is set to examine the infected area, but the product fails to inspect possibly infected objects. As an example, in the Virus Research Unit's antivirus scanner analysis 1997 one product did not detect macro viruses unless the appropriate file name extension was specifically added to the list of examined objects (Helenius 1997). In case the detection fails because the infection possibility is not known to the product, the incorrect operation belongs to the category discussed at the beginning of this section.

4.2 Antivirus product virus detection analysis

We can now combine our previous findings and examine how antivirus product virus detection analysis can be carried out. Figure 5 demonstrates how different procedures of computer antivirus product's virus detection analysis can be accomplished.

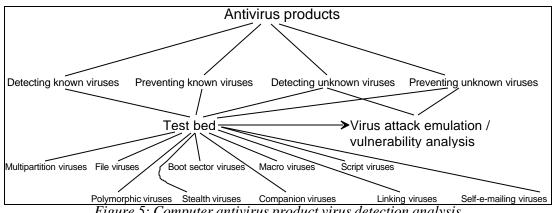


Figure 5: Computer antivirus product virus detection analysis

Each product type requires different analysis approaches. A virus test bed (see Appendix 1) can be used for evaluating products which will detect or prevent known viruses. A virus test bed can be utilised for products which will detect or prevent unknown viruses, but vulnerability analysis (see Appendix 1) is also required. If the viruses in the test bed are divided into different categories, this can be utilised while analysing antivirus products. The different virus categories of the test bed are examples and the classification can be different depending on the analysis methods and products evaluated. If the test bed is divided into different categories, this will help analysis of products.

Antivirus product category	Current antivirus products representing the category
Detecting known viruses:	known virus scanners
Preventing known viruses:	memory resident known virus scanners
Detecting unknown viruses:	checksum calculation programs and heuristic scanners
Preventing unknown viruses:	memory resident heuristic scanners, behaviour blockers and memory resident checksum calculation programs

Table 4: Current antivirus products

The previous finding discussed in Subsection 4.1.1 suggests that antivirus products can be divided into four different categories. Current³ antivirus products representing each category are in Table 4. Different types of computer antivirus product require different virus detection analysis approaches.

In the following subsections we will discuss different categories of Figure 5 and Table 4. We will briefly present definitions of some techniques current

³ In this work we define the word current to mean the time of completing this dissertation, which is May 2002.

antivirus products are using for virus detection and prevention. We will also summarise the disadvantages and advantages of different types of product.

4.2.1 Virus scanners

A scanner tries to find known viruses by detecting their instruction sequences or unknown viruses by recognising a pattern of instructions typical for viruses. The latter approach is called heuristic scanning and the previous approach is called known virus scanning.

4.2.1.1 Known virus scanners

Known virus scanners can identify viruses and therefore a user as well as an antivirus support person can easily find out how the virus behaves. Furthermore, the known virus scanners can be combined with virus disinfection. (Further details on virus scanning techniques can be found in Muttik 2000)

A disadvantage of known virus scanners is that they need to be frequently updated and this causes resource usage for both to the user, who needs to update and test updates, as well as to the producer, who must produce the updates. Another disadvantage of known virus scanners is that the scanners can only detect viruses known to the scanner and therefore newly created viruses cannot be detected unless they resemble some already existing virus.

When the first antivirus scanners appeared, they were using only signature scanning methods. Signature scanning means that from inspected objects a scanner searches for a sequence or sequences of bytes that are present in a known virus. This is an ideal approach as long as the sequences can be chosen in such a way that they can be found in all appearances of a virus and the sequences do not exist in objects which do not contain a virus. Unfortunately, this is not always the case, but by correctly selecting long enough sequences from correct positions the possibility for false positives will be marginal. Unfortunately, this is true only for viruses, whose appearance is always the same.

When polymorphic viruses appeared finding a good enough sequence became difficult and sometimes even impossible, because such a sequence simply did not exist in all appearances of a virus. For some polymorphic viruses, signature scanning could still be easily applied, because the encryption routine that took care of the polymorphism (sometimes called encryption engine) was constant and long enough not to cause false positives. However, the author of V2P2-viruses wrote these viruses to show that signature scanning cannot be applied to all polymorphic viruses (Solomon 1994, pp. 13-18) the constant part of one variant of the virus was only two bytes long. Antivirus scanner producers had to implement other solutions.

The most advanced solution so far is called a polymorphic emulator. A polymorphic emulator emulates the encryption engine of a polymorphic virus and decrypts the content of the virus into a readable form and tries to find the virus from the decrypted form.

Scanners do not try to find viruses only from files and boot areas; they also search for known viruses in central memory in order to prevent stealth viruses from making the scanner to perceive changed objects as unchanged. Otherwise a virus active in the memory could infect those objects which the scanner investigates.

4.2.1.2 Heuristic scanners

Heuristic scanners try to find viruses by searching for virus specific behaviour in possibly infected objects (For more details on heuristic scanning, see Veldman 1995; Bontchev 1998, pp.126-135 and Ször 2000). For example, if a program contains a routine for replication, the program remains resident in memory and the program contains a hard disk formatting routine, the program probably carries a destructive virus.

The advantages of heuristic scanning are that unknown viruses can be detected and there is no need for frequent updates. The disadvantages are that heuristic scanning can be circumvented and therefore heuristic scanners are not able to detect all unknown viruses. Furthermore, a user should be able to correctly interpret the results of heuristic scanning. Heuristic scanners typically inform that suspected behaviour has been found in the searched object and often it is up to the user to decide whether this behaviour is viral or not. From this follows that false positive analysis is important for heuristic scanners. As can be expected, there are differences in the implementation of heuristic scanning in different antivirus products. Some products have been built to use heuristic mode always enabled and for other products the heuristic scanning is partially optional. We use the word partial here, because according to my experiential knowledge each current antivirus product uses some generic virus detection methods that are always enabled. For example, some products detect unknown boot sector viruses even when heuristic mode is not enabled.

Although heuristic scanning improves virus detection, the possibility for false alarms increases, too. It can be assumed that the sensitivity for false alarms is low when the heuristic scanning is always enabled, because in this case heuristic scanning is designed for every day use and information about false positives quickly reaches the producer and thus problems will be quickly fixed. Many products have different levels of heuristic sensitivity and the sensitivity for false alarms can clearly grow when heuristic sensitivity increases. This can be observed with some products simply by selecting the highest level of heuristic sensitivity and scanning the contents of a hard disk. The heuristic scanning engine quite probably generates some information about suspicious behaviour in some innocent files. It is obvious that evaluating only the virus detection part of such heuristic scanning which will cause a lot of false alarms, will indicate false sense of security, because this kind of scanning is unusable for such users that do not have enough technical understanding of the computer systems they are using.

We can conclude that the evaluation of sensitivity to false alarms becomes especially important when such parts or operating modes of products are analysed which are not used as default and which may increase sensitivity to false alarms.

4.2.1.3 Memory resident known virus scanners

Memory resident scanners are active in the computer's memory and try to catch the virus before it gets a chance to infect. The advantage of a memory resident known virus scanner is that it catches a virus before it replicates. The disadvantages are resource consumption, possible compatibility problems and the same disadvantages as with non-preventing known virus scanners.

It would be incorrect to assume that each different type of scanner from the same producer detects exactly the same viruses. Because memory resident scanners are typically implemented to use as little resources as possible, they may detect fewer viruses than normal scanners. Fortunately, memory resident scanners implemented for Windows environment are typically almost always able to prevent viruses as well as the normal identifying (non-preventing) scanner of the same product (see, for example, Helenius 1996b, 1997 and 1999a).

4.2.1.4 Memory resident heuristic scanning

As well as known virus scanning can be memory resident so also heuristic scanning can be implemented as memory resident. Although currently there do not seem to exist pure memory resident heuristic scanners, this technology does exist and is in use integrated with other types of memory resident antivirus programs, like known virus memory resident scanners and behaviour blockers. Accurately implemented heuristic scanning programs can efficiently reduce risks in certain tasks and prevent virus infections. For example, an antivirus program can be watching e-mail attachments and prevent opening attachments that contain suspicious executable code. Other advantages and disadvantages of memory resident heuristic scanning are the same as with non-preventing heuristic scanning.

4.2.2 Integrity checkers

While infecting, a virus will evidently cause changes to a system. Integrity checking can be applied to detect these changes. Integrity checking means methods to verify that specified system areas have not changed. Typically an integrity checking program counts checksums from specified system areas. Typically checksums of executable files and boot areas are calculated. Traditional checksum calculation programs are not able to verify integrity of such macros that are embedded in document files, but there are also programs for this purpose.

The advantages of integrity checking are that the integrity of the specified areas can be verified and viruses unknown to the product can be detected without frequently updating the integrity checking program. The disadvantages of integrity checking are that the checksums need to be updated whenever contents of the executable files or macros are changed. Furthermore, if a virus has already infected the system, counting checksums of the already infected objects will not reveal the virus from the object. One problem is that a checksum calculation program cannot tell whether changes have been caused by a virus or some other reason. Furthermore, some stealth viruses are able to bypass checksum calculation programs. Therefore the computer's central memory should be clean from viruses before integrity checking is used. In other words, the computer should be cold booted from a clean boot device, like a floppy diskette. One additional problem is that a virus could also infect such objects whose integrity has not been checked. Nevertheless, correctly used checksum calculation is so far the most reliable way to ensure that a system is free from viruses.

Integrity checking could be only applied to such system areas that are not normally modified. This could include boot areas, operating system files and constant applications. The advantage of this method is that viruses will probably be caught, but checksums do not need frequent updates. The disadvantage is that some viruses may infect only those objects, whose integrity is not checked.

4.2.3 Behaviour blockers

Behaviour blocking means strategies to prevent programs from performing illegal operations during their execution. A program called behaviour blocker may utilise the strategies of memory resident heuristic scanning, but we define behaviour blocking to concern such events that are caused as a result of execution of programs. Typically current behaviour blockers warn about suspicious behaviour and a user must choose correctly whether the behaviour is allowed or not.

The advantages of behaviour blocking are that unknown viruses can be prevented and frequent updates are not needed. A major drawback is that a user should be able to interpret correctly the information a behaviour blocker gives in order to be able to correctly detect viruses and reject false positives. Furthermore, tunneling viruses are able to cheat behaviour blockers in operating systems that do not have memory protection. The reason for this is that typically the system monitoring occurs after a virus has had its chance to master the system and therefore the virus is able to use methods to prevent the behaviour blocker from working correctly.

4.2.4 Memory resident integrity checkers

An integrity checking program can also be actively working in the background of a computer. The advantage of memory resident integrity checking is that changes can be detected immediately. In the same manner as with behaviour blockers, a tunneling virus may cheat memory resident integrity checking.

4.3 Antivirus product virus detection analysis methods

Now we have discussed the current methods antivirus products are using for virus detection. Next we will discuss the different antivirus product types based on Subsection 4.1.1 and presented in Figure 5. We will examine the possible virus detection analysis methods for each antivirus product type. We will not in detail discuss vulnerability analysis methods specific for current products; Bontchev has discussed known attacks against integrity checkers (1998, pp. 204-219 and 343-347), behaviour blockers (1998, pp. 200-204 and 363-365) and scanners (1998, pp. 192-200, 328-343 and 347-363).

4.3.1 Detecting known viruses

A well maintained virus test bed, which contains viruses known to computer antivirus researchers can be used for evaluating products which will detect known viruses (see, for example, Bontchev 1993). The virus detection analysis can be carried out by scanning the contents of the test bed and concluding results from the scanning reports. Unfortunately, some products may crash during the scanning and in such case the files causing crashes need to be traced and files resulting in crashes should be treated as unidentified by the product. Of course, the files need to be verified as containing viruses before they can be treated as unidentified by the product.

4.3.2 Preventing known viruses

A well maintained virus test bed containing viruses known to antivirus researchers can also be used for evaluating products preventing known viruses (see, for example, Helenius 1996b, 1997 and 1999a). The difference is that the product is working in the background and this requires more complicated evaluation methods, but the same virus test bed can be used with products, which will prevent known viruses.

4.3.3 Detecting unknown viruses

A virus test bed can also be used as a basis for the analysis for products, which detect unknown viruses. Often products detecting unknown viruses are combined with products which will detect known viruses. If possible, the products' known virus detection capability should be disabled. Known virus detection may be detached by removing virus database files, by using old database files or by using specific operation mode of a product. Unfortunately, the known virus detection may be an inseparable part of a product and in this case the test bed should be limited to viruses not known to the product and a vulnerability analysis (see Appendix 1) may be necessary.

As discussed in Section 1.6, the creation of new viruses is considered as unethical among computer antivirus professionals. Different types of virus attack should therefore be emulated instead of writing or creating new viruses. The effects of a virus could be emulated and a product's capabilities against these attacks could be evaluated. The effects should correspond to different existing virus types. The analysis could also take in account effects which current viruses do not have, but which could be potential virus attack methods.

4.3.4 Preventing unknown viruses

A virus test bed can be also used for evaluating products which will prevent unknown viruses. The difference is that the product is working in the background and this requires special evaluation methods, but the same virus test bed can be used with products which will prevent unknown viruses. This is demonstrated in Virus Research Unit's behaviour blocker analysis (see Helenius 1999a). With products preventing unknown viruses, virus attack emulation and vulnerability analysis are also required.

4.4 Different virus types in the test bed

As presented in Figure 5, the test bed can be divided into different categories and some viruses may require some special analysis methods depending on the product type evaluated. We will next discuss different virus types and their influence on analysis methods.

4.4.1 File viruses

File viruses as well as other virus types must be capable of spreading further and so must also the replicates be capable of spreading further. Virus detection analysis can be carried out by scanning or processing files in the way required for a particular product type.

4.4.2 Boot sector viruses

For file viruses the analysis process can be simple, but for boot sector viruses some special arrangements are required. Manually feeding hundreds or thousands of infected diskettes and repeating the cycle several times is too time-consuming and frustrating.

Many scanners have an option to scan boot images written on files. Unfortunately, scanning for boot sector viruses in files does not correspond to a real user situation and is likely to cause errors. According to our experiment a scanner sometimes behaves differently when a true working virus on a floppy diskette is scanned. Gryaznov (1994) has programmed a suitable tool for DOS-scanners called Simboot. Simboot emulates infected floppy diskettes by writing infected diskette images to memory and by assigning a memory segment as a floppy diskette drive. Simboot is a fast and suitable for DOS-scanners, but if memory resident scanners or other types of product need to be analysed, some special technology is required.

Another way to solve the problem is to have image files (see Appendix 1) of infected floppy disks on a hard disk or on a network server. The images can be written one by one on suitable floppy diskettes. Each image can then be scanned or accessed one by one. In this way the analysis process can be automatic.

4.4.3 Macro viruses

Like traditional binary viruses macro viruses must be capable of spreading further and the replicates of macro viruses must be capable of spreading further. An additional Windows related problem is that memory resident scanners for Windows and Internet environments must be capable of finding macro viruses before they get a chance to replicate further. From this it follows that the tasks a user could perform in Windows must be emulated.

4.4.4 Script viruses

Script viruses should be replicated by using the environment needed for replication. For example, viruses using MS-DOS batch language should be replicated using batch files as goat files (see Appendix 1) and viruses using Visual Basic Scripting should be replicated using Windows Scripting Host.

4.4.5 Multipartition viruses

Ideally multipartition viruses should be replicated on each type of object they are capable of infecting. Then each of the objects should be analysed separately. The reason for this is that it may sometimes happen that a product is capable of finding a virus in one type of object but incapable of finding the virus in other types of object. For example, viruses which replicate on both files and boot areas should be replicated on both types of objects.

4.4.6 Polymorphic viruses

Polymorphic viruses try to mislead virus scanners by varying appearance. Therefore it is essential to generate several different samples of each polymorphic virus. There is a possibility for an antivirus scanner to miss part or all of the replicates of a virus even when a scanner can find the original virus sample. Therefore a test bed should include several replicates of the original virus sample infected on different files. There is no absolute truth regarding the correct number of replicates, but in general the more replicates are generated and used, the better is the estimate for the correct detection capability. However, the more replicates are created and used the more time and resources analysis processes will take. The optimal number of replicates also depends on the virus type. For polymorphic viruses it may be a necessity to have several hundred or thousand replicates to estimate the correct detection rate.

Moreover, a non-polymorphic virus can be replicated to several different hosts and this is even preferred. It is possible that a product's identification data is incorrect and therefore it can find the original sample, but not the replicates or only part of the replicates. Furthermore, when a virus infects different types of objects (for example, different types of executable files with varying file name extensions, document files and boot sectors), an antivirus program may be able to find the virus only in certain types of object.

The creation of several replicates does not ensure that the detection results reflect on antivirus product's actual detection capabilities, but the probability of a correct estimate increases. The importance of new replication can be observed, for example, by trying replication of some viruses received from an antivirus vendor. Sometimes a product can detect viruses in the original sample files, but not all of the replicates.

4.4.7 Companion viruses

Companion viruses sustaining known executable appearance do not pose much difficulty for scanners, because they can be simply detected by normally scanning executable files. Companion viruses, however, may mislead nonidentifying products, like integrity checkers, if the possibility of a companion virus type of attack has not been taken into account while implementing the product.

4.4.8 Stealth viruses

Stealth viruses try to hide the changes they have committed in a system. In order to efficiently do this a stealth virus actively stays in the background. Sometimes it happens, that an antivirus product can find a virus when the virus is not active, but the same product may not find the virus when the virus is active on the system. However, those products which detect viruses before the virus gets its chance to control the system should not be analysed when the virus is active in the memory.

In the worst case the product might be actually replicating the virus, because the virus could infect each executable file the product opens for reading. Therefore it would be ideal to perform stealth virus detection analysis when the virus has been activated. Unfortunately this kind of approach is tedious and therefore unlikely to be performed in antivirus scanner analyses. Obviously, making the process automatic could facilitate the analysis and as demonstrated in the Virus Research Unit's antivirus scanner analyses 1999, 1997 and 1996 (Helenius 1999a, 1997 and 1996b), this is feasible.

4.4.9 Linking viruses

Linking viruses may require that the system is first infected with the virus in order to construct the linkage. However, scanners typically detect the virus even when the linkage does not exist and this can be utilised in virus detection analysis. Furthermore, a linking virus may be capable of replicating even without establishing the linkage, but if this is not the case, then the linkage should be created before analysis. Otherwise we are not analysing true working viruses, because the virus is not capable of replicating without the linkage.

4.4.10 Memory resident viruses

As demonstrated with the definition of stealth viruses, memory resident viruses may be able to deceive antivirus products, if the memory scanning does not work correctly for some reason and the virus active in the central memory is not found. In such a case it is possible that an antivirus scanner is actually replicating a virus, because the virus may infect each file the scanner opens for reading. Therefore one phase of antivirus product evaluation could be evaluating antivirus products' capabilities to detect viruses in central memory.

4.4.11 Self-distributing viruses

Self-distributing viruses have at least one special replication channel from a local system to a remote system. The replication should be performed by using the replication channels. However, the replication environment should be an isolated environment in order to prevent the virus accidentally spreading to external systems. Preventing antivirus products should be analysed based on the prevention mechanism. This may require that the replication channel is used or that the virus is activated while the antivirus product is actively preventing viruses.

4.5 Some special problems of computer antivirus product evaluation

The purpose of this dissertation is not to discuss antivirus product evaluation (see Appendix 1) in detail. However, in order to provide theoretical background of computer antivirus product evaluation, we will discuss briefly some special problems of computer antivirus product evaluation.

4.5.1 False alarms

Each type of antivirus product may give erroneous information that it has found a virus, when in reality there is no virus in the inspected object (see Section 4.2). As Trendy (1996) demonstrates, false alarms can cause much trouble and financial loss for computer users and organisations when antivirus products are trying to find and remove a virus, which never existed. A product giving a lot of false alarms may even be useless. Therefore false alarm analysis should be part of a thorough antivirus product evaluation.

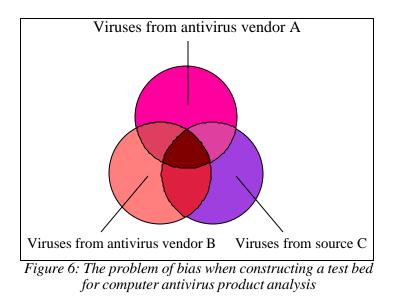
The problem with false alarm rate analysis is that it is difficult to create reliable indicators estimating products' false alarm sensitivity. The sensitivity for false alarms cannot be accurately evaluated, because it is impossible to maintain all current and future objects which may or may not cause false alarms. Such a test base would contain all possible present and future program codes for a computer system and it is evident that creating and maintaining such a collection is impossible.

One possible way to perform a false alarm rate evaluation would be to have thousands or more innocent files which would be scanned or executed, when antivirus products are analysed. This method does not give exact results, but if there is a large enough test base, a false alarm rate test could show which products or products' settings are sensitive to false alarms.

4.5.2 The problem of bias

If a person analysing antivirus products obtains viruses only from one antivirus producer and he/she uses only these viruses in his/her test bed, it is likely that the producer's antivirus program will detect all the viruses. Thus the test bed will not indicate correct results, even if the analysis otherwise is correctly performed. This is actually one general problem of large test beds. As long as there are more and more viruses, no-one can be certain of possessing all existing viruses. Moreover, even if virus production were to stop, even then no-one could be sure that he or she had all the existing viruses, because some viruses may not exist among the virus sources used. Thus a test bed representing all viruses is always more or less biased. The level of bias depends on how many different virus sources an analyser has used for preparing the test bed.

Figure 6 illustrates the problem of bias. Each source group of viruses has overlapping sequences with other groups, but also a unique set of viruses which do not exist in other groups. Now, suppose that a test bed did not have viruses from antivirus vendor A, but antivirus vendor A's product was still included in the analysis. The test bed would be favouring antivirus vendor B and penalising antivirus vendor A. The test bed should contain viruses from both antivirus Vendor A and antivirus vendor B. In addition, the virus test bed should contain other known viruses appropriate for the test bed, like viruses from source C.



The problem of bias also emphasises the importance of re-infecting viruses on new goat files (see Appendix 1) and performing the analysis only with viruses found in the field. However, it can be argued that other viruses also do constitute a threat. One can use a large test bed, but then he or she should regularly receive viruses from all those antivirus vendors whose products are included in the analysis. I am convinced that this condition has not always been fulfilled with antivirus evaluations using large virus test beds.

4.5.3 Viruses found in the field versus laboratory viruses

There is a huge difference in the number of viruses which are found in the field and the number of so-called laboratory viruses, which have been discovered only in virus collections. This can be observed, for example, by viewing the Wildlist Organisation International's (originally Joe Well's) list of PC viruses in the wild (The WildList Organization International 1993-2002) and comparing the number of viruses on the list with the number of viruses that antivirus products promise to find. Even if we suppose that the Wildlist does not contain all viruses found in the field and that the number of viruses that products promise to find also contains non-viruses, still less than 10% of all viruses are found in the field. It is more likely that computer users will deal with the 10% of viruses in the wild than with the 90% of laboratory viruses. This finding supports the idea that antivirus scanner evaluations should contain an "in the wild" test bed containing viruses found in the field. However, constructing the "in the wild" test bed is not as obvious a task as it may seem. For a virus to be included in the test bed, it must have been found in the field at least once. This is not, however, as obvious as it sounds. Before a virus can be included in the test bed, someone must have reported to some antivirus researcher that the virus has been found in the field, but how do we know that someone has reported the virus to some anti-virus researcher?

One possible solution is to use the Wildlist Organisation International's wildlist, which includes viruses that have been reported as being found in the field according to antivirus researchers accepted as reporters. The list does not, however, contain all the viruses found in the field, because not all the cases are reported to the list maintainers. For example, in Finland there have been viruses found in the field, which have been reported to antivirus researchers and/or to the Central Criminal Police, but some of them never appeared on the wildlist. I also know on the basis of reports from other anti-virus researchers, that viruses have been found in the field which have not been on the wildlist. However, the Wildlist Organisation International's wildlist is so far the only reliably formed list covering most of the antivirus companies and it is also recognised as a valid source of information by most computer antivirus researchers.

Furthermore, Bontchev (1999) has discovered some problems with the wildlist. He mentions such problems as interrelation dependency of the wildlist reporters; only viruses causing problems are reported, mostly new viruses are reported, samples of viruses are not sent and therefore such reports are ignored. Bontchev also found some methodological errors such as problems with data gathering, lack of clear definitions, sluggishness, classification problems and virus naming problems.

Most of the 'in the wild' lists do not have exact information such as which variants of viruses were found in the field. Sometimes the exact variant can be identified directly, but in most cases further examination is needed. This causes problems when constructing the test bed. Sometimes the original virus can be received from antivirus researchers but unfortunately this is not always possible. Several sources of information have to be compared to determine which variant of the virus has been found in the field. In most cases this process of comparison will produce results and the correct variant can almost certainly be identified, but still there may be cases when the variant is not chosen correctly.

One more problem with the "in the wild" test bed is that it is possible to affect the test results by single incidents or simply by lying. For example, a producer could claim that he has found a new virus from some user's computer and other products would not be able to find that particular virus. This argument indicates that unclear cases should be sent for comment and then removed unless there is other evidence or only those cases should be included which have been reported by at least two different antivirus researchers. Then, of course, all possible viruses found in the field cannot be included, but the most spread viruses will be included. For example, I have used the method of including only clear incidents in the Virus Research Unit's antivirus scanner analyses based on Joe Well's list 3/1996 (Helenius 1996b), 7/1997 (Helenius 1997) and 10/98 (Helenius 1999a). Only the first part of the Joe Well's list was used to construct the test bed.

4.5.4 Determining the potential threat posed by each virus

It would be idealistic to perform the evaluation in such a way that the potential threat caused by each virus could be measured. Some viruses are more widespread than others and thus are more likely to be encountered than those which have not spread as widely. Unfortunately, there does not currently seem to exist a valid basis for determining the threat. The wildlist could be used as a measurement basis but as it reads on the list: "The list should not be considered a list of the most common viruses, however, since no specific provision is made for a commonness factor." (The WildList Organization International 1993-2002). There do exist some automatic software solutions that measure the distribution of viruses in some areas of the Internet like Dr Solomon's Virus Patrol and Trend Micro World Virus Tracking Center (Trend 2001), but these are still limited metrics. Dr Solomon's Virus Patrol reports only viruses sent to news groups and is thus limited to e-mailing viruses and certain kind of virus attacks. Moreover, Trend Micro World Virus Tracking Center collects information only from its own products and does not distinguish different virus variants accurately. Obviously there is a need for valid and frequently updated information measuring commonness of different viruses.

In addition to the commonness of a virus, an important issue is also the potential that each virus possesses for successful replication. For example, if a virus works with only some rarely used operating system or application, large incidents are unlikely to occur. Furthermore, such qualities of a virus as the replication mechanism, visibility and compatibility affect the likelihood of a virus to replicate. Unfortunately, the potential of a virus to spread efficiently is also difficult to estimate. However, a carefully constructed classification scheme estimating replication potential could be possible and useful.

4.5.5 The threat of unknown viruses

Viruses that are known to exist are not the only problem; unknown and forthcoming viruses also cause threat. This is most essential for quickly spreading self-distributing viruses, because these viruses may replicate globally within a day and should therefore be discovered and prevented quickly. Unfortunately, it is not possible to estimate accurately what kind of viruses will appear in the future.

However, it is possible to utilise the techniques of known viruses. Of course, as discussed in Subsection 1.6, an ethical antivirus researcher should not create viruses by himself and therefore investigating the threat of unknown viruses should be investigated by trying to reveal weaknesses in antivirus products by other means. Existing viruses which are not known to a product could be used, but it may sometimes be difficult to find such viruses. Different virus specific techniques could also be applied without actually writing a virus. For example, vulnerability analysis of checksum calculation programs could be realised by simulating different attack methods viruses could use.

4.5.6 The ever growing number of different viruses

The number of different viruses has been growing very quickly since the first viruses were found. During the first years the growth was almost exponential (see, for example, Kephart et al. 1991). The increasing growth is evidently a problem to antivirus developers (Skulason 1994), but it is also a problem for those who wish to analyse antivirus products. Each new virus will cause extra work for an antivirus reviewer because replicates of the virus must be included in the test bed and more product evaluation time is needed for the new replicates. This problem also supports the idea of automatic virus replication. Because there are so many viruses, the only realistic way to verify that all viruses are true working viruses is to use some automatic virus replication mechanism or increase manpower.

4.5.7 Regular access to antivirus products

One problem is that for an adequate comparison the antivirus products analysed should be from the same time period. This means that an antivirus analyst should have regular access to antivirus products. He or she should contact antivirus producers or their representatives directly and agree on regular updates. There are updates and evaluation versions also available from the Internet. A special time limit for sending a product should be avoided, because otherwise it is possible to affect evaluation results. Antivirus producers might publish their update just before the time limit or even provide a special version for evaluators. Such a version could perform well in detection analysis, but the disadvantage could be increased sensitivity for false alarms. We can conclude that instead of giving an exact time limit, it would be better to have regular access to the updates and the producers should not be aware of the exact time limit. The evaluation process should not affect the antivirus production process.

4.5.8 Reliability problems

One problem with antivirus scanner evaluations is that it is too often unclear how antivirus evaluators are performing the tests and what viruses they are using in their tests. If only the detection percentages or final results of an antivirus evaluation are published, there is no way to verify the results. Unfortunately, antivirus reviewers often give only little information or no information at all of their test bed or test methods. Test methods and test bed should be so accurately reported that the tests can be repeated and verified, if necessary. (This requirement is similar to that presented to a researcher studying a science and obeying the most ideal rules.) Detailed information about test bed and test methods should be available to at least those antivirus product producers, whose products were included in the analysis.

Antivirus product evaluation is demanding and it is easy to make mistakes. Therefore before publishing, evaluation reports should be sent for comment to all those antivirus producers whose products were included in the analysis. This is also a matter of fairness, because in this way each antivirus producer has a chance to point out possible weaknesses in the test bed or in the evaluation process before publishing final results and the evaluator can correct the mistakes and thus avoid publishing misleading information. The verification does not, however, necessarily reveal all possible weaknesses in the test bed or in the test bed or in the evaluation process. Therefore a person evaluating antivirus products should always critically reflect on his or her own work.

4.5.9 Replicates of viruses

Sometimes antivirus products can detect only part of the replicates (see, for example, cross-references of the antivirus scanner analyses 1994-1999 (Helenius 1994a, 1995b, 1996b, 1997 and 1999a). This is especially true in the case of polymorphic viruses. Furthermore, sometimes it happens that an antivirus scanner can detect only the virus in the original file, but not the replicates of the virus. The probable reason for the failed detection in this case is that the antivirus developer has failed to include a correct signature or detection method of the virus. Therefore it is important to replicate the virus to new objects.

4.5.10 Ensuring that each virus in a test bed is a true working virus

Antivirus products should be built to find true working viruses, which are capable of replicating recursively by themselves. After all, true working viruses are what computer users are likely to face. Therefore Trojan horses, joke programs, intended viruses, first generation viruses, innocent files, damaged files (see Chapter 3 for an explanation of these terms) and other non-viruses should be excluded from the virus test bed (as demonstrated in Bontchev 1993). After all, in a virus detection analysis we should analyse how well products can detect viruses.

Figure 7 illustrates program code classes to exclude from a virus test bed. However, Trojan horses, malicious toolkits and joke programs can be included, if other malicious program code than viruses need to be used. Then each program type should be in a separate test bed. The reason for this is the difference of each program code type. Trojan horses, malicious toolkits and joke programs are not likely to spread as efficiently as viruses. In addition, joke programs are not likely to cause as much harm as viruses or Trojan horses.

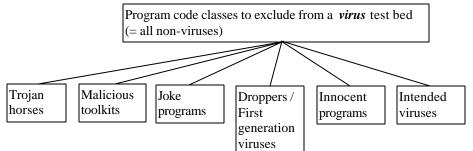


Figure 7: Program code classes to exclude from a virus test bed

The non-virus removal process is perhaps the most difficult task of antivirus product analysis. It is not enough to ensure that all non-viruses are removed from the test bed. The original files might be sent to antivirus vendors, and it might be, that some products can detect only the original files or part of the replicates, but not all replicates of the virus. Therefore each original file should be replicated further for at least two generations.

We must prove that an object contains a virus. Otherwise we will take a chance that we will make a mistake. Figure 8 illustrates the reason for using at least two generations for the proof. The original files may include viruses, but also Trojan horses, joke programs, intended viruses, damaged files, droppers and even completely innocent files. After the replication process the second generation includes only viruses or damaged files. The third generation is needed to find out which files of the second generation are damaged and which files are capable of spreading further. The replicates of the second generation can be reserved for use in later analyses and for verification purposes.

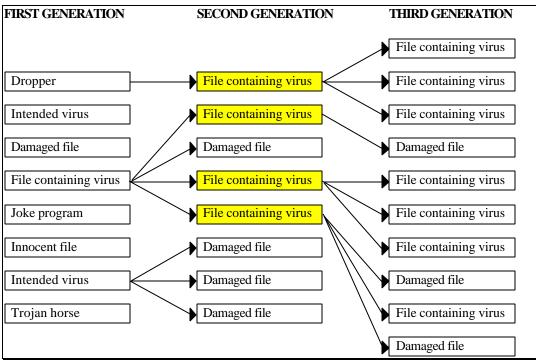


Figure 8: Replicates of suspect files

4.5.11 Differentiating virus variants

A virus is considered to differ from some other virus, if the constant codes of the viruses differ from each other by at least one byte. The term "constant code" is used here in order to demonstrate that different replication generations of a polymorphic virus represent the same virus.

It does not matter whether the difference is in the executable virus code or in the virus data area. The virus data area contains some data that moves with the virus, but is not executed. An example of virus data is a text message that a virus prints on the screen. Please note that the data area does not need to be used by the virus. It is enough that the virus carries the data area. One problem with differentiating virus variants is that a virus is often attached to a host and the virus code must be separated from the host before the exact binary area of the virus can be examined. The separation can be accomplished by manual analysis or, if the host is known, it is easy to separate the virus code. However, in the case of polymorphic viruses it is not enough to compare the binary form of the virus. A manual analysis of the virus code may be required. (Further details on manual analysis can be found in Bontchev 1998, pp. 222-229.)

Another problem is that an object may be infected by several different viruses. In this case manual analysis may be required. (Further details on this matter can be found in Bontchev 1998, p. 253)

Known virus scanners can be used as a valuable help in differentiating virus variants. If some scanner reports a different virus variant in some sample files, then there is good reason to assume that the sample files do indeed contain different viruses. Furthermore, some scanners can notify, if a host is infected with more than one virus. However, scanners may provide false information and therefore the difference might be explained by some other reason. For example, in the case of a polymorphic virus, a scanner may give erroneous information in some appearances of a virus.

5. Development of computer-supported methods for computer antivirus product virus detection analysis

We have now introduced a variety of virus detection and we will next discuss how automatic virus replication as well as other automatic virus detection evaluation tasks can be implemented. By automatic we mean that a once initiated task can be carried through without human intervention.

We will discuss in this Chapter such design problems that have been interesting and technically demanding. Design problems with a simple solution have been omitted and only the final outcome of a problem has been described. We will next present a system called Automatic and Controlled Virus Code Execution System, which was implemented at the Virus Research Unit.

5.1 Development phases of the Automatic and Controlled Virus Code Execution System

The Automatic and Controlled Virus Code Execution System was initially developed for automatic virus replication in a controlled environment. Automatic virus replication means that without human intervention we can replicate viruses to a desired set of objects and store the infected objects. The benefit of automatic virus replication is that we can replicate viruses to desired objects with minimal human effort and control.

We will next start to describe the development of the system in chronological order. We will start from the initial idea and proceed to describe the development of automatic file virus replication, extension to boot sector virus replication and macro virus replication. We will present different alternatives that were found and argument the decisions to proceed.

Figure 9 illustrates the development phases of the Automatic and Controlled Virus Code Execution System. A rectangle signifies an automated procedure. The time order is from top down and an arrow signifies that a new method was developed from a previously developed method. On the left side are those processes that were target states and on the right side are those processes that were invented as a side effect. In other words, the target state was not defined at the beginning. It appeared that the system could be easily applied to other tasks and there was a need to automate the task. Then a new target state was set. A bold rectangle means a developed method and a thin rectangle means a function of developed processes.

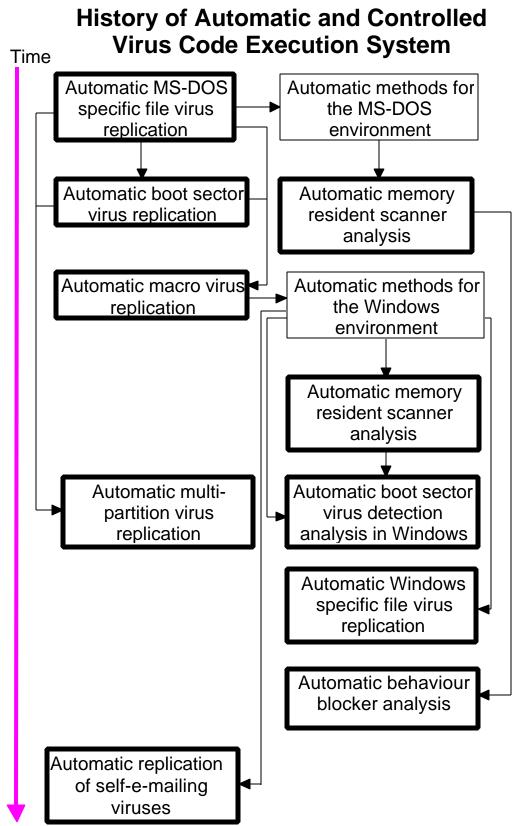


Figure 9: Development phases of Automatic and Controlled Virus Code Execution System

5.2 The initial idea

It all began at the beginning of the year 1994 after noticing that a requirement to replicate viruses is a necessity for computer antivirus product evaluation. Automatic virus replication is needed for constructing a virus test bed and verifying that the virus test bed contains viruses. As can be concluded from Subsections 4.2, 4.3, 4.4 and 4.5 the construction of a virus test bed is a basic condition for antivirus product virus detection analysis. From this observation it followed that the first version of computer-supported virus replication was constructed. However, at first there were alternatives from which we had to select.

5.2.1 Manual or computer-supported virus replication?

At first it was necessary to decide how the replication could be handled. The alternatives were either a manual replication or automated replication process.

Manual replication process had the following properties:

- Could be applied immediately
- Requires continual efforts
- Allows more experiential learning of viruses than automatic replication method

Automatic replication process had the following properties:

- Requires development effort
- Cannot be applied immediately although early stages of the replication system could be used while the development continues
- When completed only maintaining and enhancement efforts will be required
- Less experiential learning of viruses is required

The automatic replication method was chosen because it was considered to save human work effort on the long run.

5.2.2 Make or buy?

The subsequent question was, how to implement the system. The alternatives were to develop it ourselves or ask others to develop the system. Since financial resource was limited and there seemed to be no other applicable systems available, the only choice was to develop the replication system ourselves.

5.2.3 Emulation or hardware implementation?

Then the next question was, how to implement the system. Two possible choices were found and the following properties were found for the choices.

Emulator of the operating system and replication process executed on the emulated system:

- Requires only one computer
- Can be run on several separate computers as separate processes
- Success of the replication process depends on the quality of the emulator
- Difficult to extend for several operating systems or boot sector virus replication
- Development requires learning accurate technical details of the operating system
- Requires long development process before the emulator can be used

Hardware implementation:

- Allows successful replication
- Allows flexible future development
- Requires several computers and customised hardware
- No need to learn accurate technical details of the operating system
- Can be used while still incomplete and development can continue parallel with the usage

The hardware implementation was chosen because it was assumed to allow successful and flexible future development. Furthermore, hardware implementation could be applied quickly for virus replication.

5.3 First specification

Next, a general specification for the automatic file virus replication was developed without knowing how every detail of the system would be implemented. Figure 10 presents the structure of the first specification of the file virus replication system.

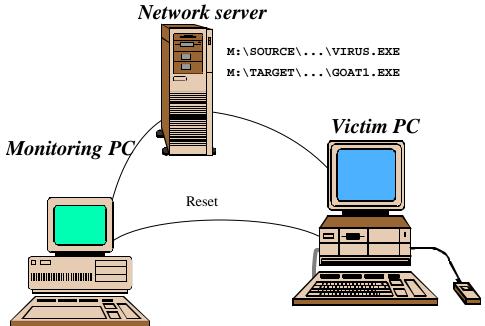


Figure 10: Structure of the first specification of the file virus replication system

The idea was that there would be three computers. One would be called "Victim PC" meaning that viruses would be executed on that computer. To achieve a safe boot after virus code had been executed there was a need for external control. It was decided that the controlling would be implemented by using an external computer and it was called as "Monitoring PC". The hardware had to be modified so that the Monitoring PC would be able to boot the Victim PC.

The Victim PC would contain certain goat files (see Appendix 1) that would be automatically executed after a floppy diskette boot. After this virus infection process the Monitoring PC could cold boot (see Appendix 1) the Victim PC from a write-protected floppy diskette. After the cold boot the system memory would be free from viruses and the computer could safely log into the network server, which would be attached to the Victim PC. Next the Victim PC could perform an infection analysis of the system and copy infected objects to the network server. Finally, the Victim PC could inform the Monitoring PC via the network connection that a new infection process would start and the Monitoring PC could start a new process. The network server was a vital component of the system, because it was needed for storing files to be processed, infection analysis, recovery and communication between the Victim PC and the Monitoring PC. For recovery image files (see Appendix 1) were stored on the network server and programs stored on the network server could be utilised when the boot was clean from viruses.

5.4 First implementation

The computer-supported processes were constructed by starting from a simple implementation and gradually extending to more complex implementations. When one stage was discovered to be working more features and new processes were constructed. The first implementation was constructed from a simple partially working implementation to a stable solution.

5.4.1 Simple implementation

The first stage of the system was prepared without realising the automatic external boot and therefore the Monitoring PC was not yet needed. The Victim PC always booted from a write-protected floppy diskette and there was a control file on the hard disk. The control file's existence determined whether the Victim PC logged into the network and began infection analysis and automatic recovery or the Victim PC started file virus replication process. The file virus replication was implemented by a batch file, which executed the infected file, goat files and other programs on the computer. The virus infection analysis was implemented by using a checksum calculation program. Changed files were copied to the network server. The recovery was implemented by writing a bit to bit image of the clean hard disk to an image file (see Appendix 1). The image file could then be used for recovering a clean system by writing contents of the file back to the hard disk. Booting was implemented by a simple program, which called system cold boot. However, the system could not operate on its own for very long, because often viruses halted the system and manual cold boot was required.

5.4.2 Automated cold boot

The next obvious enhancement was to implement the external boot control. This was implemented by short-circuiting the computer's reset signal by external control. For the external control the Monitoring PC was now needed. Changing a certain bit of the Monitoring PC's parallel port changed the Victim PC's reset signal. Now the external boot control made file virus replication completely automatic excluding CMOS-memory failures (see Appendix 1). CMOS-memory failures could not be handled, because there was no boot device available, if the floppy diskette drive was not correctly defined in the computer's CMOS memory settings.

The Monitoring PC had a program that checked whether a certain file appeared in a network directory where the Victim PC had write rights. If the file did not appear or if it could not be removed within a certain beforehand-programmed time, the Monitoring PC automatically booted Victim PC.

5.4.3 Improved replication

While the replication was made automatic, some viruses could still not be replicated. Therefore different processes were needed to improve the replication success. As the Victim PC used a Novell network, the system could not replicate viruses that did not work, if a network driver was installed. Therefore a process was needed where network connection was not available. Furthermore, some viruses required a certain operating system and certain types of file to infect before infection could occur. For example, some viruses may not infect self-prepared goat files (for more details, see Muttik 1995). The solution was to implement three different processes with different operating system versions with varying system settings and goat files. In addition to self-prepared goat files were executed as a user would execute them.

5.5 Automatic boot sector virus replication

Now file viruses could be replicated automatically, but boot sector viruses could not be processed. The next step was to modify the system to also handle boot sector viruses.

5.5.1 Boot device selection

There was a need for an additional boot device and automatic boot device selection, because only in this way could boot sector viruses be replicated. When boot sector viruses are replicated, the Victim PC needs to be booted first from a possibly infected floppy diskette after which the hard disk may have become infected. Then some clean boot device is needed because clean diskettes must be placed on floppy diskette drives as a user would change the diskettes. After successful hard disk infection the Victim PC needs to be booted from the hard disk. Finally, the Victim PC needs to be booted again from the clean boot device so that infection analysis and recovery can be accomplished.

The additional boot device was realised by installing a network card that had network boot ROM installed. The network card's boot ROM allowed the Victim PC to boot from the network. A network boot means that a computer can be booted from the network by using a boot image file installed on the network server.

5.5.1.1 Selecting a boot device from the network card

A network card with boot ROM requires user input for selection of a boot from the hard disk or from the network. The network boot ROM used required a user to press the letter H before the boot process, if a boot from the hard disk was required. If the letter H was not pressed, boot continued from the network. From this it followed that there was a need to control keyboard externally.

The problem was solved by implementing a device which a computer could use to control another computer's keyboard. The device was called a keyboardcontrolling device. The first implementation of the keyboard-controlling device was developed externally.

5.5.1.2 Selecting a boot from a floppy diskette

There was still one major problem, because there was the need to boot from a floppy diskette in addition to booting from the network and from the hard disk. If a floppy diskette was in the diskette drive, then the boot would start from the floppy diskette. Otherwise control was moved to the boot ROM, but automated boot from all boot devices could not be achieved. The solution was to implement additional hardware customisation by switching the electricity of the floppy diskette drive off for a few seconds during the boot process, if a boot from the floppy diskette was not desired. Thus control was shifted to the boot ROM even when a floppy diskette was in the diskette drive. Then whether the boot started from the hard disk or from the network was a matter of selection.

5.5.1.3 Selecting a boot from 3.5 or 5.25-inch floppy diskette

Finally, it was realised that an enhancement would be to apply the same computer for replicating viruses from both 3.5 and 5.25-inch diskettes to both types of diskette. Therefore the floppy diskette drive order would need to be changed whenever necessary. This was accomplished by swapping the cable connection by external control. However, in order to get the floppy diskette drives to work correctly the CMOS memory settings had to be changed accordingly whenever the drives were swapped. This could be accomplished by writing correct CMOS memory settings into the CMOS memory before a new replication process.

5.5.2 The implementation

As all the required boot devices could be automatically selected, it was possible to implement automatic boot sector virus replication. Furthermore, CMOS memory failures could be handled, because the boot ROM did not require CMOS settings to be correct. A boot from the network was always possible with the help of the keyboard-controlling device, even if the CMOS settings were incorrect. After realising access to the network server correct settings could be written back to the CMOS memory.

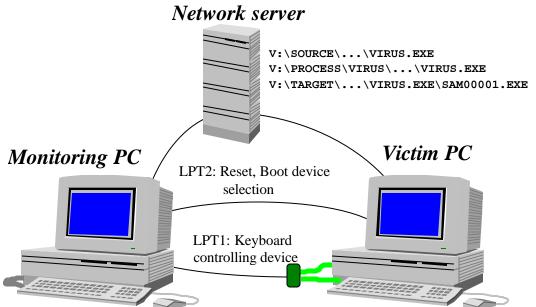


Figure 11: The structure of the system after implementing keyboard controlling device and boot device selection. The Victim PC and the Monitoring PC are attached to the network server, which is used for storing infected objects. The Monitoring PC controls the Victim PC's keyboard and boot operations.

Now the components of the system appeared as in Figure 11. The hardware improvements implemented were keyboard controlling and automatic boot device selection. Figure 12 presents how boot sector viruses were processed.

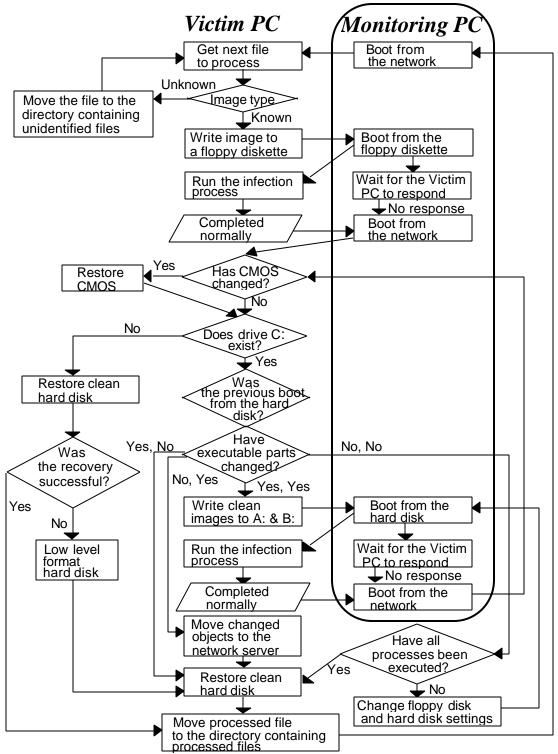


Figure 12: Automatic replication process of boot sector viruses. A rectangle means a state of a computer, a diamond means a choice and an arrow means the proceeding direction from one stage to another.

We will next discuss the replication process presented in Figure 12. Before preparing a new floppy diskette image file to be processed, the system examined from which type of diskette the image file was obtained. This was initially accomplished by checking the file name extension. Certain file name extensions were made as decision signals to a program determining the diskette format. If the file name extension was not one of those originally determined, a program written especially for this purpose automatically checked content of the image file. If the diskette type could not still be identified, the file was moved to a directory containing unidentified boot sector images (in the implementation the directory was V:\PROCESS\UNKNOWN).

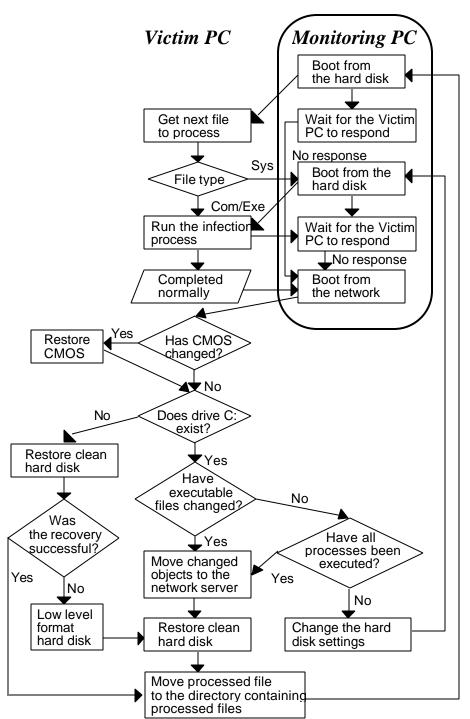
If the diskette type was recognised, the image file was written to a correct type of diskette and the Victim PC was booted either from 3.5 inch or 5.25 inch floppy diskette drive depending on the image type. When the program on the diskette's boot sector asked a user to remove the diskette and press any key to continue, the keyboard controlling device was used for pressing a key and booting continued from the hard disk when the Monitoring PC temporarily switched the floppy diskette drives off. When the replication process was completed or the Victim PC was not responding, the Monitoring PC cold booted the Victim PC from the network server.

The next thing to do was to check what had been changed on the Victim PC. If the CMOS memory's content had been changed, the original content was restored by writing the original content back to the CMOS memory. If the hard disk was not present, original content was restored from the image file stored on the network server, and if this did not help, the hard disk was low level formatted. If no change was detected on the hard disk, the next boot sector file was processed. If changes were detected, the changed files were moved to the directory containing changed objects (V:\TARGET) and the clean contents were stored on floppy diskettes and the Victim PC was booted from the hard disk. At first the Victim PC attached diskettes simply with the DIR command, but if this did not help, a more thorough process was executed, which copied files, attached diskettes various ways, executed files from various locations etc. If this did not help, the clean diskette types were changed from high-density diskettes to double density diskettes.

It was later noticed that some viruses cause the hard disk to be accessible only when the virus is resident in central memory. Therefore the replication process was changed in such a way that if the hard disk was not present, the process would still continue. Furthermore, it was noticed that the floppy diskette drives worked properly only if CMOS memory settings were set corresponding to the floppy diskette drive order. Therefore the process was modified so that whenever floppy diskette drive order was changed CMOS memory settings were also changed.

5.6 Improved file virus replication

The network boot and keyboard-controlling device also enabled some improvements for the file virus replication process. The floppy diskette boot was no longer needed, because the replication process could be started by a boot from the hard disk. This was a faster and more successful replication method. Furthermore, CMOS memory failures could also be automatically



fixed and handled with the file virus replication process. Figure 13 presents how file viruses were processed.

Figure 13: Automatic replication process of file viruses. A rectangle indicates a state of a computer, a diamond indicates a choice and an arrow indicates direction from one state to another.

When file viruses were processed, the system picked up files one by one from a network directory containing files to be processed (in the implementation the directory was V:\SOURCE). First of all, the Victim PC was booted from the clean hard disk and next the possibly infected file to process was obtained from the network directory. If the file obtained was a SYS file, it was installed as a

device driver and the computer was booted from the hard disk. Otherwise the replication process started directly. The Victim PC informed the Monitoring PC that virus code was about to be executed and the Monitoring PC started to check whether the Victim PC was responding normally or not.

When the virus code execution and the goat file execution had been completed normally or the Victim PC was not responding, the Victim PC was automatically cold booted from the network. Precautions had been taken against a malicious program booting the Victim PC during the infection trial. If this happened, the booting continued from the network while a virus could remain resident in the central memory. To ensure that the network boot was always clean, the Victim PC could not continue operations from the network unless the Monitoring PC had authorised the operations.

After booting from the network the Victim PC started to check possible changes in the system. If the CMOS memory's content had been changed, the boot process was continued using the keyboard controlling device and the original CMOS memory was automatically recovered.

The next thing to check was whether the hard disk was present any more or not. If the hard disk was not present, the original hard disk image was restored from the network server and the original process file was moved to a directory tree containing files which had caused hard disk errors (V:\PROCESS\DEL). If restoring from the original image did not help, the hard disk was automatically low level formatted.

If the hard disk was present, the next thing to do was to check whether executable files had been changed or not. If changes were detected, changed files were moved to the directory tree containing changed areas (V:\TARGET), the original process file was moved to a directory containing files that had caused changes in files (V:\PROCESS\VIRUS), the clean hard disk was restored and the process could continue.

If no changes were detected or changes were detected only on partition or boot sector, a more thorough process was executed. This was accomplished by changing system date and time, by trying a different set of goat files, by executing DOS commands, by changing the Victim PC's configuration and by changing the operating system version. The operating system version was only changed, if partition sector or boot sector of the hard disk was not changed. The reason for this was that the virus was likely to replicate without operating system change, if boot areas were changed.

5.7 Automated processes for the MS-DOS environment

While the system was designed only for automatic virus replication, as a side effect it was discovered that the system could also be used for other tasks in the MS-DOS environment (as discussed in Helenius 1995a). We will next discuss how the memory resident scanner detection analysis was implemented in our antivirus scanner analyses (Helenius 1994a, 1995b, 1996b, 1997 and 1999a). The memory resident scanner analysis belongs to the category of analysing products preventing known viruses (see Subsections 4.2.1.4 and 4.3.2) and preventing unknown viruses (see Subsections 4.2.1.4 and 4.3.4). We will first discuss the automatic processes for analysing memory resident scanners then with boot sector viruses. We have discussed virus detection analysis of these virus categories in Subsection 4.4.

5.7.1 File virus detection analysis of memory resident scanners

We used two basic methods for virus detection analysis of file viruses. One is called the file copy method and the other file execution method. When using the file copy method, virus detection is evaluated during copying infected files and with the file execution method infected files are executed.

5.7.1.1 File copy method

It was noticed already before the system was developed that memory resident scanners' virus detection capabilities can be analysed against file viruses by copying files when the memory resident part of a product is activated. A major drawback of this file copy method is that it does not ensure that a product can prevent the same viruses also when infected files are executed or opened. However, it can be argued that if a product is developed with such consistency that it uses the same detection methods for both file copy and file execution, the file copy method gives a reliable estimation of the product's virus prevention capabilities. The file copy method can be automated by creating a large batch file, which will copy each object to a destination directory. The batch file can be created from a list of files in the test bed by using some text editor with macro support, or for more convenience, a special program can be written for this purpose. Figure 14 presents an example of a batch file created by such a special tool program.

DELTREE C:\MISS
MD C:\MISS
MD C:\MISS\XM
MD C:\MISS\XM\LAROUX
MD C:\MISS\XM\LAROUX\A
COPY M:\MACRO\XM\LAROUX\A\HML.XLS C:\MISS\XM\LAROUX\A\HML.XLS
COPY M:\MACRO\XM\LAROUX\A\MATDID1.XLS C:\MISS\XM\LAROUX\A\MATDID1.XLS
COPY M:\MACRO\XM\LAROUX\A\MATDID2.XLS C:\MISS\XM\LAROUX\A\MATDID2.XLS
COPY M:\MACRO\XM\LAROUX\A\OPETSUUN.XLS C:\MISS\XM\LAROUX\A\OPETSUUN.XLS
COPY M:\MACRO\XM\LAROUX\A\PELI.XLS C:\MISS\XM\LAROUX\A\PELI.XLS
COPY M:\MACRO\XM\LAROUX\A\PELI2.XLS C:\MISS\XM\LAROUX\A\PELI2.XLS
COPY M:\MACRO\XM\LAROUX\A\RUUDUKKO.XLS C:\MISS\XM\LAROUX\A\RUUDUKKO.XLS
COPY M:\MACRO\XM\LAROUX\A\TESTI3.XLS C:\MISS\XM\LAROUX\A\TESTI3.XLS
MD C:\MISS\X97M
MD C:\MISS\X97M\LAROUX
MD C:\MISS\X97M\LAROUX\A
COPY M:\MACRO\X97M\LAROUX\A\MATDID1.XLS C:\MISS\X97M\LAROUX\A\MATDID1.XLS
COPY M:\MACRO\X97M\LAROUX\A\HML.XLS C:\MISS\X97M\LAROUX\A\HML.XLS
COPY M:\MACRO\X97M\LAROUX\A\MATDID2.XLS C:\MISS\X97M\LAROUX\A\MATDID2.XLS
COPY M:\MACRO\X97M\LAROUX\A\OPETSUUN.XLS C:\MISS\X97M\LAROUX\A\OPETSUUN.XLS
COPY M:\MACRO\X97M\LAROUX\A\PELI.XLS C:\MISS\X97M\LAROUX\A\PELI.XLS
COPY M:\MACRO\X97M\LAROUX\A\PELI2.XLS C:\MISS\X97M\LAROUX\A\PELI2.XLS
COPY M:\MACRO\X97M\LAROUX\A\RUUDUKKO.XLS C:\MISS\X97M\LAROUX\A\RUUDUKKO.XLS
COPY M:\MACRO\X97M\LAROUX\A\TESTI3.XLS C:\MISS\X97M\LAROUX\A\TESTI3.XLS
Eigene 14. Content of a batch file which coming a number files of two views of

Figure 14: Content of a batch file, which copies sample files of two viruses

The batch file copies sample files of two Excel macro viruses from a directory tree to the destination directory C:\MISS. After the execution of the batch file the destination directory includes viruses missed by a memory resident scanner working in the background. A batch file copying all viruses from a test bed would of course, be much larger, containing thousands of rows.

Solving the user action problem

There is an additional problem involved with memory resident scanners. When a scanner finds a virus, it will typically display a warning message and require a user to take action before the batch file can copy the next file. There are at least three possible approaches for solving this problem. One is to have a memory resident utility, which will press the required key corresponding to a correct user action and this approach has been used from the beginning in the Virus Research Unit's antivirus scanner evaluations. The development of the Automatic and Controlled Virus Code Execution System also provided another solution for the problem. The external keyboardcontrolling device can be used for the desired user action. The device can control the computer's keyboard and press the required key or keypress sequence. The first implementation of the keyboard-controlling device was applicable only for MS-DOS products, but the enhanced keyboard-controlling device is not dependent on the operating environment.

Sometimes the required key combination can be settled simply by putting weight on the desired keys. This presumes that the required key press sequence is simple, the program does not require a key to be lifted and pressed again and the keyboard buffer will not become overloaded.

5.7.1.2 File execution method

Sometimes memory resident scanners are able to detect more viruses when an actual virus code is being executed. For example, Dr. Solomon's Antivirus Toolkit, IBM Antivirus and McAfee Scan were able to detect more viruses when files infected with viruses were executed instead of copying infected files. Unfortunately, automating file execution method is a more complicated and a slower method than the previously discussed file copy method and requires a customised hardware solution. However, the Automatic and Controlled Virus Code Execution System also enabled the file execution method and it was used with the Virus Research Unit's antivirus scanner analyses 1996, 1997 and 1999 (Helenius 1996b, 1997 and 1999).

The problem with the file execution method is that when a product has failed to detect a virus, the system may have become infected. Therefore each time a virus escapes, the system must be recovered. Fortunately, the recovery can be accomplished when the PC can be automatically cold booted externally. Then the clean boot required for safe recovery can be achieved either by a boot from the network or by a boot from a write-protected floppy diskette. After a clean boot the system recovery and infection analysis can be safely performed. Although automation of the file execution is entirely feasible, a major drawback is that it is much slower than the file copy method. The file copy method is multiple times faster because simply pressing the required key combination can process a virus sample.

Bontchev (1998, p. 241) describes an alternative approach for testing memory resident scanners with file execution method. The idea is based on using a memory resident utility program that will prevent execution of the infected file after a scanner has failed to intercept a virus. The drawback of this method is that it does not work with scanners that aim to bypass tunneling viruses.

5.7.2 Boot sector virus detection analysis of memory resident scanners

Boot sector virus tests of memory resident scanners can be carried out by writing images of infected diskettes on floppy diskettes one after another and by attaching infected diskettes. Now the same problem as with file virus detection of memory resident scanners arises. Whenever a scanner finds a virus, it will typically display a warning message and require a user action before it will allow the operation to be continued. The solution for this problem can be the same as with file viruses.

Simboot (presented in Subsection 4.4.2) can be also used for analysing memory resident scanners. Simboot must be repeatedly launched from a batch file with a different configuration file for each floppy boot image. Instead of launching a scanner, a virtual diskette can be accessed by Simboot's configuration file. Again caution should be exercised, if Simboot is used.

5.7.3 Automatic virus detection analysis of MS-DOS behaviour blockers

After implementing the file execution method for memory resident scanners, it was noticed that the same principle could also be easily applied to virus detection analysis of behaviour blocker programs for MS-DOS with a file virus test bed. This method was used with the Virus Research Unit's antivirus scanner analysis 1999 (Helenius 1999a). This virus detection analysis method belong to the category of analysing products preventing unknown viruses as suggested in Subsection 4.2.

The principle was the same as with memory resident scanners with file execution method. If a virus could cause change in some areas of the system, this was interpreted as a failure of a behaviour blocker to prevent a virus infection. Like the file execution method, this method is also time consuming because of the need to continually boot and recover the Victim PC.

The results of the first behaviour blocker analyses were only experimental, they concerned only file viruses and a thorough behaviour blocker analysis would include a vulnerability analysis (see Appendix 1). It would be more complicated to also include boot sector and macro viruses although these processes seem to be possible.

The same principle could also be applied to behaviour blockers compatible with Microsoft Windows. However, the process would probably be so time consuming that it could not be applied without optimising and establishing parallel processes. The process would be slower because of increased system loading and recovery time. The process could probably be applicable, if the recovery and boot frequency could be optimised. The recovery could be optimised by replacing only changed system areas instead of recovering the whole content of the hard disk. Boot frequency could be optimised by booting the Victim PC only when a virus had escaped.

5.8 Multipartition virus replication of file and boot sector viruses

During the antivirus product analyses conducted at the Virus Research Unit I noticed that some file viruses require floppy diskette access before the virus can replicate. Therefore the boot sector virus replication process was extended in such a way that it could start the replication process from an executable file. If the processed file was an executable file instead of a boot sector image, the file was executed in the same way as with file virus replication, but floppy diskette access was a major part of the replication process. Accordingly, if the original sample file was a floppy diskette, executable files were also accessed, executed and copied on both hard disk and floppy diskette. In this way multipartition viruses infecting both floppy diskettes and executable files could be replicated. The virus detection analysis of multipartition viruses was discussed in Subsection 4.4.5.

5.9 Automatic macro virus replication

In September 1995 the first macro virus, called WordMacro.Concept or in short WM.Concept, appeared (Virus Bulletin 1995). The virus spread rapidly around the world and a new virus creation boom followed based on application macros (see for example, virus prevalence tables in the Virus Bulletin Journal from August 1995 to December 1998). After different macro viruses began to appear at increasing speed, it was vital to extend automatic replication to macro viruses. Initially the replication was implemented to work with Word for Office 95, but thereafter the replication has been implemented for other Word macro virus types and different Excel macro viruses. We will first present the ideas that were used with Word for Office 95 and then we will present how macro virus replication tasks were extended to other types of macro viruses.

5.9.1 Solving the user action problem

Because replication of a macro virus typically presumed that a user performed certain actions with some Windows application, automatic macro virus replication required that Windows environment should be externally controllable. Two possible alternatives were seen. The one was to implement the control by a program working in the background of Windows environment and the other was to implement the control by external computer controlling the keyboard. At that moment the main function of the keyboard-controlling device of the Automatic and Controlled Virus Code Execution System was to control the Victim PC's keyboard by programs executed from the Monitoring PC. The first implementation of the keyboard-controlling device was not appropriate for Windows environment. It was implemented by a direct connection from the parallel port to the keyboard. The success of controlling the keyboard was dependent on the system speed of the computers and it behaved differently in different applications. In addition, there was a controlling relay, which caused disturbance. The controlling device was still

reliable for simple keyboard sequences in applications for MS-DOS, but it did not work in Windows environment.

The following two possible choices for automatic keyboard controlling were found:

A program working in the background

- Does not need customised hardware
- Vulnerable in case of system failures
- Requires programming efforts
- Difficult to transfer for various operating environments

Improved keyboard controlling device

- Requires development effort
- Requires customised hardware
- Once implemented, the device is not dependent on the operating environment
- Can be easily applied to other tasks besides virus replication
- Enables a parallel system for the current system. The new system can be utilised automating tasks in Windows environment

The improved keyboard-controlling device was chosen, because it made possible to implement parallel system for Windows environment and it was not dependent on the operating environment. The term 'operating environment' is used here instead of 'operating system' to illustrate that the keyboardcontrolling device can be used in any environment where keyboard input is needed. For example, the device can be used for controlling a computer's CMOS memory settings even before the operating system is loaded.

The implementation was realised by using a keyboard and connecting electric circuits inside the keyboard in such a way that the keyboard sent correct signals corresponding to different key presses. The implemented device was now as reliable as a keyboard can be. Next the control programs were written in such a flexible way that the keyboard controlling could be easily applied for different purposes simply by writing different script files. Now, as the Monitoring PC could use the keyboard-controlling device for emulating user actions, the automatic macro virus replication could be implemented. Figure 15 presents the operations of the Victim PC and the Monitoring PC during automatic macro virus replication.

5.9.2 Implemented replication process

We now have a basic idea of the components of the system and next we will present how the system performs automatic macro virus replication. One important function of the system is to carry out automatic virus replication. This is required for creating test files for a virus test bed (see Appendix 1) and for verifying that the virus test bed contains only viruses capable of replicating further (see Subsection 4.5.10).

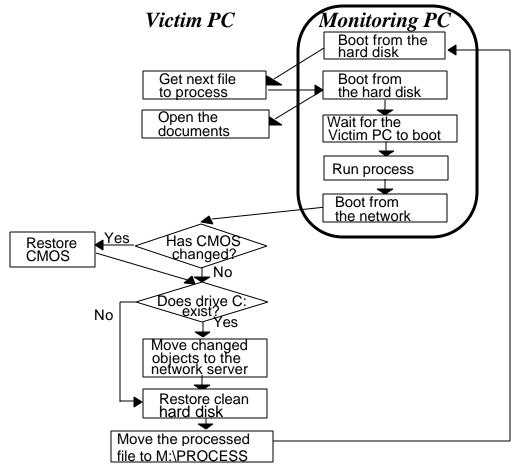


Figure 15: Operations of the Victim PC and the Monitoring PC during automatic macro virus replication. A rectangle indicates a state of a computer, a diamond indicates a choice and an arrow indicates direction.

When the macro virus replication was started, the Victim PC was booted from a clean hard disk. The Victim PC established a connection to the network server with read only rights and picked up the source document from a network directory containing source sample files. The source path and file name could, for example, be M:\SOURCE\WM\CONCEPT\A\VIRUS.DOC. The document file was then copied to the startup directory and to Word's default directory for document files. Target documents were also copied to corresponding directories and the Victim PC was booted again from the hard disk. Because the documents were in the startup directory, they were automatically opened. The Monitoring PC waited until it could be certain that the Victim PC had loaded the operating environment and opened the document files.

Now the Monitoring PC took advantage of the enhanced keyboard-controlling device and started using menus of Word for Windows. To accomplish replication, the Victim PC performed several things which were typical infection methods for macro viruses. This included such as closing files, opening files from different locations, saving files in different directories using the "Save" and "Save As" selections from the file menu, switching between different documents, closing Word, starting Word again and repeating the operations. Different replication operations could have been chosen by inspecting the macros from the infected document, but it seems that the current⁴ solution works with current macro viruses and therefore a general replication process was used as a replication mechanism.

After the Monitoring PC had carried out all the replication operations, it shut down the Victim PC and booted it from the network. Now boot from the network was clean and thus the Victim PC could perform all required operations safely. First the Victim PC checked for changes in traditional executable files or boot areas. If there were changes, the possibly infected objects were copied to the network server.

Next there was the problem of determining which documents had been infected. Traditional integrity checking could not be applied, because a document file's content will change each time it is saved. Therefore it was decided to write a special utility program for the macro virus replication system. The program checked Word document files and wrote a log file of each file containing macros. The log file was then used for determining which documents could be infected by a virus and possibly infected documents were copied to the network server.

Next the system was recovered. The original hard disk was restored from the image file and the system was again clean from viruses. The original sample file was copied to a directory corresponding to success of the replication. Finally, the Monitoring PC reset the Victim PC and the next document file could be processed.

⁴ In this work we define the word current to mean the time of completing this dissertation, which is May 2002.

5.9.3 Other replication environments for macro viruses

After implementing the replication for Office 95 version of Word, it was noticed that other types of macro viruses had also become common. At that moment this meant Office 97 macro viruses and Excel macro viruses. Because the keyboard controlling was implemented to be flexible, other replication environments could be controlled simply by writing new scripts for the controlling program and changing hard disk settings. However, the program searching macros needed to be updated in order to be able to detect macros in different file formats.

5.10 Automating other tasks in Windows environment

We have now presented the principles of automatic macro virus replication but, as stated previously, the flexibility of the extended system made it possible to automate other processes in Windows environment. These processes were constructed after macro virus replication and we will next discuss these. These processes are automatic replication of file viruses designed for Windows, boot sector virus detection analysis in Windows environment and virus detection analysis of memory resident scanners for Windows environment. We will also present other possible tasks that can be utilised.

5.10.1 Automatic replication of file viruses for Windows

The improved keyboard-controlling device was also suitable for automatic replication of viruses infecting Windows executables. Because the system was designed to be flexible, the replication mechanism was easy to realise. The replication was simply realised by using the keyboard-controlling device for starting and closing Windows programs. Traditional checksum calculation could be applied for the infection analysis.

5.10.2 Boot sector virus detection analysis in Windows environment

Formerly it was a problem to perform boot sector virus detection analysis in Windows environment (Helenius 1996b, p 7; Helenius 1995b, p. 8; Helenius 1994a). After implementing the improved keyboard controlling device even boot sector virus analysis could be automated because controlling Windows via the keyboard was possible. In the Virus Research Unit's antivirus scanner analysis 1997 boot sector virus detection analysis of Windows 95 scanners was performed by writing diskette images one by one on floppy diskettes and by launching the scanning from the graphical environment (Helenius 1997, p 9). The keyboard-controlling device was utilised for automating this task. All analysed products made it possible to launch the scanning by using the keyboard.

With boot sector viruses there were, however, some special problems. One problem was that Windows 95 typically does not notice that a diskette's boot sector has been changed unless the diskette has been physically removed from the floppy diskette drive. Using the Monitoring PC to switch the power of the floppy diskette drive first off and then back on solved the problem.

Another problem was observed when we were preparing the antivirus scanner analysis 1997. After negotiations with computer antivirus product producers, I was notified that Windows 95 corrupts some boot sector viruses and therefore some scanners cannot detect some of the corrupted viruses. These scanners may, however, be able to detect actual working viruses. The solution for the problem was to add one more customisation to the system. The diskette image writing operation was performed in MS-DOS mode and the diskette was physically write protected in Windows. A drawback of this method is that it slows down the boot sector virus tests, because the Victim PC must switch frequently between Windows and MS-DOS mode. Nevertheless, this method was used in the Virus Research Unit's Antivirus Scanner Analysis 1999 (Helenius 1999a).

5.10.3 Memory resident scanners for Windows environment

It was quickly realised that memory resident scanners for Windows environment can also be analysed by creating a batch file for copying infected files. The target directory tree stores missed sample files and the principle is the same as with MS-DOS scanners (see Subsection 5.7.1). The keyboardcontrolling device can be used for closing the dialog a scanner produces whenever it finds an infected file.

5.10.4 Automatic replication of self-e-mailing viruses

For the self e-mailing virus replication e-mail service was a necessity for the system. The e-mail service was realised by installing a Debian Linux (see Debian 2001) server running sendmail service.

In the Victim PC there was Microsoft Outlook e-mail program and Windows Scripting Host installed. The e-mail program's address book contained such email addresses that the e-mail server could deliver. The Victim PC opened the suspicious file. If the suspicious file was part of an e-mail message, it was opened using Microsoft Outlook. Next the Victim PC was used to emulate a real user usage of the system and finally e-mail was sent by using the keyboard-controlling device for controlling Microsoft Outlook. Next the Victim PC was restarted and the operations were repeated. The e-mails received needed to be processed automatically. The access to the server was established in a clean stage of the Victim PC by Lan Manager boot from MS-DOS. A floppy diskette was used for the Lan Manager boot, but our intention is to replace the network card with such a card, which can be used for a network boot from the Linux server. After the Lan Manager boot the Linux server could be accessed in MS-DOS mode. Now e-mail folders could be accessed and such e-mails that contained attachments were stored on the network server. Furthermore, changed executable files and document files containing macros were stored. An additional adjustment was that changed Windows registry files and Windows initialise files were recorded. In addition, extra files that appeared on the system were observed and stored. Finally, a clean hard disk was recovered from an image file stored on the network server.

Although at the end of writing this dissertation, the replication of self-e-mailing viruses is still under development and has not been used for published virus detection analyses, the results obtained are promising. The current process is constructed for Windows 95 and Microsoft Outlook 97. However, the replication process is easy to transform for different operating systems and e-mail programs. The transformation can be realised by changing the system configuration and writing appropriate script files for controlling the Victim PC's keyboard.

5.11 Other possible tasks

I have so far only described such applications for which the system has been used, but there are also other tasks which can be automated by utilising the system. In general this includes all such tasks which require systematic automation and which can be automated by controlling the keyboard and boot device selection. These tasks may facilitate all areas of antivirus product virus detection analysis discussed in Chapter 4. This includes, for instance, such tasks as analysing how well antivirus products can prevent viruses from spreading in different ways. For example, documents can be opened in different ways, or a computer could be infected by a virus before an antivirus product is used. The system could also be used for assessing how well antivirus products can prevent viruses coming from the Internet. For example, antivirus products should prevent viruses coming via e-mail attachments, ftp or World Wide Web.

6. Self-assessment of the computer-supported processes

Now we have presented development phases of the Automatic and Controlled Virus Code Execution System and in that way we have proved that the system can be built. The subsequent question is: How good is the system? Since we do not have accurate technical details of other competing systems or methods we will concentrate on comparing the system developed with manual processes that are required for the same tasks. We think that the main advantage of automation is time saving, and therefore we compare performance times.

Please note that we are here not discussing all implemented computersupported processes, but only such virus replication processes that we have considered the most important and from which we have gathered valuable data while using the system. We will at first briefly discuss other systems known to the author, which are capable of automatic virus code execution.

6.1 Comparison with other computer-supported systems

Previously published studies of automatic virus code execution have mostly concentrated on the MS-DOS environment and on the execution of program files. I would like to emphasise that the Automatic and Controlled Virus Code Execution System was developed without knowledge of the development of the other systems. The comparison is here included to clarify what other systems have been developed and made public parallel to our present system.

At the EICAR Conference in 1995 Swimmer discussed the Virus Intrusion Detection Expert System (1995), which was developed at the Virus Test Center at the University of Hamburg. The system was based on emulation of the 8086processor. The emulation was developed in the Unix environment. Much of the functionality of the system was based on the functionality of the emulation. According to Swimmer, the emulation did not have the capability for Windows emulation and it could not handle certain stealth and tunneling viruses. The advantage of the system was that it made possible the trace of the virus code at the machine instruction level and this was also the main objective of the system.

At the Virus Bulletin Conference in 1995 Leitold presented the Automatic Virus Analyser System (1995), which was based on a public bulletin board system. The system had some elements in common with the Automatic and Controlled Virus Code Execution System. It had a Slave PC and a Master PC corresponding to the Victim PC and the Monitoring PC of the Automatic and Controlled Virus Code Execution System. The Master PC could reset and change the floppy diskette drives of the Slave PC, but there was no network or device for controlling the keyboard. Therefore the system lacked many important properties of the Automatic and Controlled Virus Code Execution System.

At the same Virus Bulletin Conference 1995 Aubrey-Jones discussed automatic testing of memory resident antivirus software (1995). He presented a system consisting of a Control PC, a Test PC, hardware links between these computers and the network server. Even network booting of the Test PC was implemented as a part of the system. As we can conclude, the system had many similarities with the Automatic and Controlled Virus Code Execution System. The main difference was that the system was designed for automatic execution of file viruses when memory resident scanners were activated. At that moment the system did not have all the possibilities of the Automatic and Controlled Virus Code Execution System, although the system had the potential to develop further. In a private e-mail conversation David Aubrey-Jones informed me that the system has now been extended to the Windows environment. Unfortunately, I have not been able to find out more about the extension.

There are also some other systems like the FRISK Software's automatic virus replication system and IBM's immune system concept (Kephart et al. 1994). According to IBM representatives IBM's system should be able to handle macro viruses as well as self-e-mailing viruses. However, IBM's system has a different approach. It has been developed for generating automatic virus signature updates and thus the function of the system is different. However, as Kephart et al. (1997) show, the system includes virus replication. Because the system was designed for commercial antivirus product development, it is difficult to find technical details about the system's implementation.

At the Virus Bulletin Conference in 2000 Whalley (2000) presented a system that allows automatic replication of self-distributing viruses. The system is based on emulating Internet in a single computer. At the moment of writing the paper the system was not yet taken into productive use. Nevertheless, according to Whalley, the system will be able to provide valuable automatic worm analysis features.

Now we have discussed other systems known to the author. As we can see, there seems to be no other systems for automatic virus replication or for automating the Windows environment discussed in depth. This does not mean that there are no such systems, but it seems that such systems have not so far been discussed in such detail neither any system for experimental use so that we could make a precise comparison. For this reason we will concentrate on comparing the computer-supported methods developed with manual methods. Comparison with manual methods was chosen also because, as can be observed from Subsection 5.2.1, they were a real alternative for automated methods.

6.2 Comparison with manual virus replication processes

The difference between manual and automated virus replication processes is that manual processes do not utilise customised hardware implementations to automate all required operations. However, in manual virus replication process a human may sometimes accomplish more sophisticated solutions for incidental situations. The argument for assessing the manual processes by myself is that I am an expert in the research area and therefore I am able to construct the test conditions as correctly as possible.

We will next present results of experimental manual virus replication processes, results from automated virus replication processes and then compare the results. Our hypothesis is that automated processes are more efficient than manual processes (see Mason 1988 for theory of experimentation). The replication speed of manual methods was recorded by using a program that recorded the process starting time and the sample file name for each replication process. The time logs of automatic methods could be gathered from log files that were created during usage of the system. The sample files for manual processes were randomly obtained from virus collections that were from the time period of constructing the automated virus replication processes. Our intention was to estimate the maximum human processing efforts and therefore the intention was not to measure the human weariness that monotonous work can cause. Therefore manual processes were carried out in short enough sequences in order to exclude weariness.

The manual processes were executed in such a way that customised hardware solutions of the system were not utilised. However, such semi-automatic tasks were included in the manual replication which did not require hardware customisation, and which were likely to be used in manual replication environments. This includes such tasks as using batch files for executing goat files, automatic recovery of the fixed disk, checksum calculation, obtaining the sample file from the network server and saving changed objects to the network server. The manual process was carried out by performing manually such tasks that the hardware customisations were able to automate. This consisted of booting the computer, selecting the boot device, switching diskettes whenever necessary and executing programs or batch files.

For example, in a file virus replication task one must first boot the computer from the hard disk, start a batch file that executes goat files (see Appendix 1) and possibly execute other files, if necessary. The person executing the replication task may notice, if the virus starts to replicate and stop processing whenever appropriate. After the replication one must boot the computer from the network and start infection analysis and recovery of the computer whenever necessary. The same computers were used for manual processes as were used with automatic processes. The argument for using the same computer was to eliminate the effects that a different computer would have caused on replication time.

6.2.1 Manual file virus replication

For automatic file virus replication we had used two computers. The first implementation of the Automatic and Controlled Virus Code Execution System was constructed with a 12 MHz 80286 computer as a Victim PC and the second implementation with a 90 MHz Pentium computer. Both of these computers were also used for automatic file virus replication. Although the 80286 computer is an old one, we decided to include it in the comparison, because we have gathered valuable data during its usage. Furthermore, our aim is to argument general conclusions that are not dependent on the efficiency of the computers or the system. We will at first examine the results from the 80286 computer and then from the Pentium computer.

6.2.1.1 Results from the Virus Research Unit's 80286 computer

From 8 replication sequences we can construct the results presented in Table 5. The average time for processing one file was 3 minutes and 45 seconds. The median value was 3 minutes 2 seconds. The smaller median value can be explained by the fact that those viruses that did not replicate at first trial needed further replication trials with different system settings.

Number of processed files	48
Average	0:03:45
Median	0:03:02
Standard deviation	0:02:23
Minimum	0:01:48
Maximum	0:14:16

Table 5: Estimation of processing time in manual file virus replication

Time	Number of processed files
1 hour	16
1 day	112
1 week	560
1 month	2500

Table 6: Estimation of the number ofprocessed sample files in manual file virusreplication

If we make an assumption that one person is capable of 7 hours' efficient work per day, we can estimate the replication progression with one person working full-time on virus replication. However, we must notice that the truth might be different, because one person probably cannot efficiently continue the same process for very long and the person would probably have other duties. However, we could make an assumption that the manual replication work could be rotated between several persons and we will continue the reasoning on this assumption. Table 6 presents the estimation for the number of executed processes (= number of processed files) with the assumption of 7 hours efficient work per day 5 days a week. The time measuring a person's capabilities is based on the standard Finnish working time. 6.2.1.2 Results from the Virus Research Unit's 90 MHz Pentium computer

In order to make a more precise comparison we decided to carry out manual replication processes also by using the Virus Research Unit's 90 MHz Pentium computer. From manual file virus replication we gained the results presented in Table 7.

Number of processed files	53
Average	0:02:59
Median	0:02:41
Standard deviation	0:00:45
Minimum	0:02:04
Maximum	0:04:58

Table 7: Estimation of processing time in
manual file virus replication

Time	Number of processed files
1 hour	20
1 day	140
1 week	700
1 month	3000

Table 8: Estimation of the number ofprocessed sample files in manual file virusreplication.

We can observe that the faster computer also enabled faster processing time. Furthermore, we can observe that standard deviation was this time smaller. An explanation for this could be that difficult samples did not appear in the set of examined samples. From the results in Table 7 we can estimate the number of processed files as presented in Table 8. The estimation is based on similar reasoning as in the case of the 80286 computer.

6.2.2 Manual boot sector virus replication

We decided also to carry out boot sector virus replication with the Pentium computer and we found the results presented in Table 9. We can observe that the standard deviation is high (4 minutes and 3 seconds). A boot sector virus sample could be processed quickly when the virus did not seem to replicate. If a boot sector virus replicated to the hard disk, time passed while preparing clean floppy diskettes, infecting floppy diskettes and storing infected floppy diskette image, it took longer to write the contents of the image file to the floppy diskette than in case of partial images.

Number of processed files	48
Average	0:07:45
Median	0:08:24
Standard deviation	0:04:03
Minimum	0:01:21
Maximum	0:20:14

Time	Number of processed files
1 hour	8
1 day	55
1 week	270
1 month	1100

Table 9: Estimation of processing time in
manual boot sector virus replication.

Table 10: Estimation of the number of processed sample files in manual boot sector virus replication.

From the average value in Table 9 we can estimate the number of processed files as presented in Table 10. The estimation is based on similar reasoning as in the case of manual file virus replication.

6.2.3 Manual macro virus replication

The same 90 MHz Pentium computer was also used for macro virus replication. We decided to perform manual macro virus replication with Windows 95 that had Microsoft Word of Microsoft Office 95 installed. The reason for this choice was that most data of automatic replication was gathered from this environment. In fact, the same system configuration was used as with automatic macro virus replication. The same system configuration was easy to achieve, because the same image file could be used.

We found the results presented in Table 11. We can observe that one process took approximately nearly 20 minutes. This can be explained partly by the fact that the hard disk infection analysis and recovery time was rather slow (7 minutes and 17 seconds). One can argue that recovery can be accomplished faster. However, our decision was to use the same recovery method as with automatic processes, because if we had a faster and reliable recovery method, we would have also applied it to automatic virus replication.

Number of processed files	43	
Average	0:19:47	
Median	0:18:58	
Standard deviation	0:02:51	
Minimum	0:16:31	
Maximum	0:29:33	
Table 11: Estimation of processing time in		

manual Word macro virus replication.

Time	Number of processed files
1 hour	3
1 day	21
1 week	110
1 month	430

Table 12: Estimation of the number of processed sample files in manual Word macro virus replication.

Again from the results in Table 11 we can estimate the number of processed files as presented in Table 12. The estimation is based on the similar reasoning as in the case of manual file virus replication.

6.2.4 Manual replication of file viruses infecting Windows executables

The results received from manual replication of file viruses infecting Windows executables are presented in Table 13. Again the same Pentium 90 MHz computer was also used for macro virus replication and Office 95 installed on Windows 95 was used because data from automatic processes was gathered from this environment.

Number of processed files	31
Average	0:12:48
Median	0:12:46
Standard deviation	0:02:44
Minimum	0:05:18
Maximum	0:18:17

Table 13: Estimation of processing time in manual replication of file viruses infecting Windows 95 executables.

Time	Number of processes
1 hour	5
1 day	35
1 week	165
1 month	670

Table 14: Estimation of the number of
processed sample files in manualWindows95 executable file virus replication

From the results we can construct the estimation of processing time presented in Table 14. The estimation is based on similar reasoning as in the case of manual file virus replication.

6.3 Automatic virus replication processes

We have now discussed the efficiency of manual replication processes and we will next examine the efficiency of automatic virus replication processes. The processes were executed by using the Automatic and Controlled Virus Code Execution System discussed in Chapter 5.

The efficiency of the automatic replication processes depends mainly on the efficiency of the Victim PC, the efficiency of the network and whether suspected viruses are replicating at the first trial or not. The efficiency of the Monitoring PC is not a critical part of the system, because control operations are fast to execute. Most of the time the Victim PC is working and the Monitoring PC is waiting for a new set of tasks.

If the replication task stopped for some reason we have excluded such cases, because our intention is to estimate optimum processing time. The processing may have halted because of a dysfunction of the system. However, more probable reasons are that the samples put for replication ran out, the network server ran out of disk space or the processing was stopped manually. Unfortunately, the log file recording was not built to report why the processing stopped.

6.3.1 Automatic file virus replication

For automatic file virus replication we had used two computers. The first implementation was constructed with a 12 MHz 80286 computer as a Victim PC and the second implementation was constructed with a 90 MHz Pentium computer. Both of these computers were used for automatic file virus replication and we will first examine the results from the 80286 computer and then from the Pentium computer.

6.3.1.1 Results from the 80286 computer

	1	-		
	Total	Replication	Replication	Replication occurred at
		occurred at	occurred at	third trial or the
		first trial	second trial	replication did not occur
Number of	4314	2215	2079	20
processed files				
Average	0:04:31	0:03:20	0:05:44	0:09:43
Median	0:04:05	0:03:13	0:05:34	0:09:28
Standard deviation	0:01:20	0:00:22	0:00:30	0:02:38
Minimum	0:02:40	0:02:40	0:04:35	0:06:49
Maximum	0:20:01	0:07:57	0:15:33	0:20:01

By analysing a log file created during usage of the system we found the results presented in Table 15.

Table 15: Estimation for the processing time of processed files with automatic file virus replication

Three different trials with different system settings were used with one replication process, if the replication did not occur. Therefore the replication speed depended on at which stage the virus started replicating. We can observe that when processing file viruses with the 80286 computer it takes about half an hour to handle 10 files, if the files are replicating at the first trial. If the viruses are not replicating at first trial, it will take about double the time since additional replication processes are required.

Time	Number of processed files
1 hour	13
1 day	320
1 week	2200
1 month	10000

Table 16: Estimation for the number of processedfiles with automatic file virus replication

From the average value of Table 15 we can estimate the number of processed files. By using the same kind of reasoning as with manual virus replication (see Subsection 6.2.1.1) we can find the estimate presented in Table 16.

6.3.1.2 Results from the Pentium computer

By analysing a log file created during usage of the system we found the results presented in Table 17.

	Total	Replication occurred at first trial	occurred at	Replication occurred at third trial or the replication did not occur
Number of processed files	561	154	364	43
Average	0:03:42	0:02:14	0:04:05	0:05:48
Median	0:03:54	0:02:05	0:03:54	0:05:23
Standard deviation	0:01:11	0:00:38	0:00:35	0:00:40
Minimum	0:01:54	0:01:54	0:03:51	0:05:09
Maximum	0:08:13	0:06:57	0:08:13	0:08:01

Table 17: Estimation for the processing time of processed files with automaticfile virus replication

We can observe that the processing is faster with the Pentium computer although this time the portion of viruses that replicated at second trial is higher.

Time	Number of processed files	
1 hour	17	
1 day	400	
1 week	2800	
1 month	12000	
Table 18: Estimation for the number of processed		

Table 18: Estimation for the number of processedfiles with automatic file virus replication

From the average value of the Table 17 we can estimate the number of processed files presented in Table 18.

6.3.2 Automatic boot sector virus replication

By analysing a log file created during usage of the system we found the results presented in Table 19. As discussed in Subsection 5.5.2, the system did not recognise floppy diskette types in all cases and the system did not try replication for such sample files. The log file also included these cases where the image file's type was not recognised and we have excluded such cases.

Number of processed files	373
Average	0:05:43
Median	0:04:08
Standard deviation	0:04:41
Minimum	0:01:01
Maximum	0:22:46

Time	Number of processed files
1 hour	10
1 day	250
1 week	1800
1 month	7500

Table 19: Estimation of processing time in automatic boot sector virus replication.

Table 20: Estimation of the number of processed sample files in automatic boot sector virus replication.

As with manual replication the standard deviation is high. This can be explained by the fact that if replication did not occur, the replication process was quickly ended, but if the virus seemed to replicate to the hard disk, the processing continued. Furthermore, writing partial sample files to the floppy diskette took more time than writing whole floppy diskette images.

6.3.3 Automatic macro virus and Windows executable virus replication

Automatic macro virus replication was constructed to perform similar operations for each sample file and therefore the processing time did not vary much. The operations were retarded because the image file containing all files of the Victim PC was large. Recovering only changed system areas could speed up the recovery operation. However, since I found reliability the main goal and the time consumed has not been a major problem, I have not yet at the time of writing this thesis implemented the optimised operations. The consumed time also depends on the operations written in the script files controlling the usage of the Victim PC.

With current configuration we found that approximately 4 samples can be processed within one hour. This can be observed from the Table 21, which is constructed from a log file recording replication processes from 7 November 1998 to 24 December 24 1998.

	Word Macro	Excel Macro	Win32
Number of processed files	1013	111	54
Average	0:15:52	0:18:11	0:14:42
Median	0:15:29	0:18:45	0:16:14
Standard deviation	0:01:21	0:02:56	0:03:22
Minimum	0:11:01	0:14:28	0:08:02
Maximum	0:28:57	0:32:58	0:19:56

Table 21: Automatic macro virus and file virus replication withMicrosoft Office 95 installed on Windows 95

The replication process was constructed to also handle Excel Macro viruses as well as Windows executable files and the results are presented in Table 21. The estimate for the number of processed Windows executables is presented in Table 22 and the estimate for processing Word document files is presented in Table 23.

Time	Number of processed files
1 hour	4
1 day	100
1 week	690
1 month	3000

Table 22: Estimation for the number of processed files with automatic Windows executable virus replication

Time	Number of processed files
1 hour	4
1 day	90
1 week	640
1 month	2700

Table 23: Estimation for the number of processed files with automatic macro virus replication

6.4 Comparison of manual and automatic file virus replication

We have now presented results from experiential manual replication processes and automatic replication processes. We can compare the results and we can find the estimate presented in Table 24. The results show that automatic replication process is able to process almost 4 times more files during a month provided that the automatic replication process continues uninterrupted and the preassumptions presented in Subsection 6.2.1.1 hold.

File virus replication with the 80286 computer				
Time	Manual replication	Automatic replication		
1 hour	16	13		
1 day	112	320		
1 week	560	2200		
1 month	2500	10000		
File virus replic	File virus replication with the Pentium computer			
Time	Manual replication	Automatic replication		
1 hour	20	17		
1 day	140	400		
1 week	700	2800		
1 month	3000	12000		
Boot sector virus replication with the 80286 computer				
Time	Manual replication	Automatic replication		
1 hour	8	10		
1 day	55	250		
1 week	270	1800		
1 month	1100	7500		
Word macro vir	rus replication with the Penn	tium computer		
Time	Manual replication	Automatic replication		
1 hour	3	4		
1 day	21	90		
1 week	110	640		
1 month	430	2700		
Windows executable virus replication with the Pentium computer				
Time	Manual replication	Automatic replication		
1 hour	5	4		
1 day	35	100		
1 week	165	690		
1 month	670	3000		

Table 24: Comparison of the manual and automatic virus replication processes

From the experiments we can conclude that a single manual replication process was quicker, if the virus did not replicate at once. The probable reasons for the shorter processing time were that the automatic replication process was not designed to work as quickly as possible but rather as reliably as possible. In addition, a human may reliably use some time-saving shortcuts for the replication process (for example, by pressing escape key during the system boot) and can continue processing immediately after execution of a task has stopped. With automatic process there is a need to use some time for synchronising the Monitoring PC with the Victim PC. Moreover, in case of file virus and boot sector virus replication altogether three different replication trials were processed, if the replication did not occur. If the replication occurred at once, the automatic method could quickly accomplish all needed operations.

Our overall conclusion is that the average processing time for one process was quicker with manual replication than with automatic replication. However, the main difference comes from the fact that manual replication cannot continue without interruption.

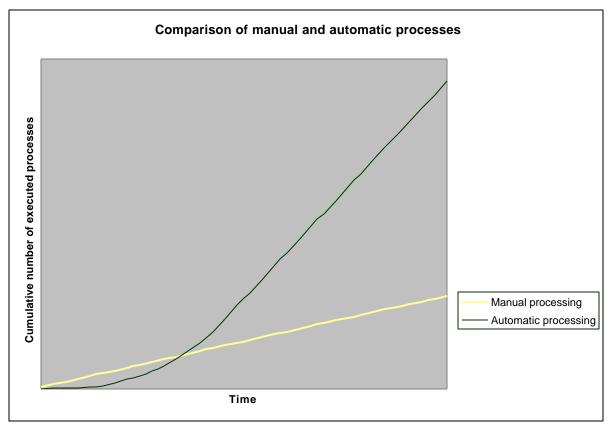


Figure 16: General comparison of manual and automatic processing

When automatic processes are developed they will require development work, but when the process is stable it will be able to execute many times more processes than manual process. As discussed in Subsection 5.2.1 the automatic process can be used during the development although not to its full extend. Our own experience of the system development suggests that in the beginning the number of executed processes is only a fraction of the executed manual processes. While the system development continues its faults and deficiencies are fixed and finally it is stable enough to work continuously and reliably. Finally, the cumulative number of executed automated processes will exceed the cumulative number of manually accomplished processes and the development time invested will be compensated as illustrated in Figure 16. In Figure 16 at the beginning the automatic process is being developed and manual processing will be more efficient, but when the automatic method can be utilised to its full extend the cumulative number of executed processes will quickly exceed the number of manual processes. The final difference between manual and automatic processes is based on the results from the previous sections.

Unfortunately, the length of the development stage or the exact number of executed processes during the development stage cannot be exactly estimated. For us it took about two months to develop the file virus replication process and about half a year to make the process stable while at the same time we were executing replication processes and doing other work duties.

We can conclude that the development of an automatic process becomes profitable, if the required number of executed processes is high or the need for processing is continuous. Furthermore, we must remember that although a human may occasionally perform tasks more quickly than an automated process, an automated system works tirelessly twenty-four hours a day. In addition, an automated system releases at least one person to do more creative and less monotonous tasks. On the other hand we must also remember that an unexpected deficiency or a sudden breakage of some essential component during usage of the automated system may take a long time to repair.

7. Discussion

We will next discuss the results and limitations of this thesis. Moreover, we will suggest possible future steps.

7.1 On the importance of the results

In Chapter 3 we presented a classification of malicious program code and established definitions. Our supplement to the classification presented by Bontchev (1998, pp. 14-112) is that we have achieved logical consistency by adding program code classes. Brunnstein (1999) presented a theoretical framework for the classification of malicious software. Brunnstein's classification is based on the concept of software dysfunctions. Although the classification presented in this thesis is easy to apply in practice as it is based on known malicious program code types. Although the classification is based on known malicious program code types, the classification is constructed in such a way that the definitions are not dependent on a particular computer system, but are of general theoretical value.

In Chapter 4 we developed the theoretical framework for computer antivirus product virus detection analysis and therefore we gathered useful theoretical information. Although many of the concepts are known, the novelty of the chapter is that we have integrated all the information and we have presented a consistent theoretical framework.

We have established a theory for the classification of antivirus products and concluded that antivirus products can be classified depending on whether products are preventing or non-preventing and whether products are identifying or non-identifying. In addition, we observed that antivirus products could fail in virus detection by false negatives, false positives and not inspecting objects on the selected system area. We then established virus detection analysis methods based on different antivirus product and virus categories.

Furthermore, we discussed the essential problems of computer antivirus product virus detection analysis and concluded that such problems need to be examined as handling false positives, the problem of bias, choosing test beds, determining the level of threat caused by different viruses, the threat of unknown viruses, the ever growing number of different viruses, reliability problems of evaluation results, ensuring that each virus in a virus test bed is a true working virus and differentiating virus variants. In Chapter 5 we discussed the construction of computer-supported processes that help antivirus product's virus detection analysis. As March and Smith (1995) suggest, an instantiation itself is a research outcome. We have discussed the development phases of computer-supported methods for computer antivirus product detection analysis and therefore proved that a system enabling these processes can be built. Furthermore, we have demonstrated in our antivirus scanner analyses (Helenius 1994a, 1995b, 1996b, 1997 and 1999a) that the computer-supported processes can efficiently be used.

We have demonstrated that the Automatic and Controlled Virus Code Execution System can be used as a powerful tool for virus replication in a controlled environment and for other tasks which require execution of virus code or controlling user interfaces via the keyboard. The system is completely automatic and it can be left to work on its own. The system and automated processes can save enormous work effort and they free resources for other tasks. Automatic virus execution is virtually essential for professional antivirus product evaluation. The system is a novel innovation as it has been developed independently without knowledge of other systems' implementations. Compared to other published systems ours has properties and functions that have not so far appeared in the other systems.

In Chapter 6 we concluded that the development of an automatic process becomes profitable, if the required number of executed processes is high or the need for processing is continuous. Furthermore, we concluded that although a human may occasionally perform tasks more quickly than an automated process, an automated system works tirelessly twenty-four hours a day. In addition, an automated system releases at least one person to do more creative and less monotonous tasks. However, we also concluded that an unexpected deficiency or a sudden breakage of some essential component during usage of the automated system might take a long time to repair. Moreover, our results and experiments suggest that manual replication is applicable for casual replication and when there is a need to analyse virus code. There can be a need to analyse virus code, for example, when we cannot make a suspected virus to replicate.

The development of the computer-supported processes required several years of innovative work and we would like to measure the goodness of the system. Pressman (2001, pp. 79-108, 507-538, 653-669) has presented metrics for software engineering. However, our finding is that traditional software metrics are difficult to apply in our case and we will examine the reasons for this.

The software in the Automatic and Controlled Virus Execution System is based on programs calling each other. Each program is a unique entity and not dependent on how other programs have been implemented as long as we know the function of each program. Some of the components are our own, some of them were externally developed and some are system tools. Furthermore, there are some programs originally developed externally and which we have improved ourselves. For most of the external components we do not have the source code available. For those software components which we have developed, the programming language used varies including such programming languages as Pascal, DOS batch language, Assembler and script language that is an outcome of development of the system. Furthermore, both in the Monitoring PC and in the Victim PC the software runs parallel. For these reasons such matters as what is the source code or even size of the system are ambiguous. Our proposal is that the only reasonable metrics are those which are not dependent on internal implementation of the programs.

The best measurement for complexity we have found is the classification Giddings (1984) presented. Giddings classified complexity of computer programs into three categories:

- 1. Domain independent software ("Software is distinguished by the independence of the problem statement and the universe of discourse.")
- 2. Experimental domain dependent software ("Software is characterised by an intrinsic uncertainty about the university of discourse. The essential problem is producing software useful for testing a hypothesis or exploring unknown characteristics of the universe.")
- 3. Embedded domain dependent software ("*The software is characterised by interdependency between the universe of discourse. The use of the software may change both the form and the substance of the universe of discourse and, as a result the nature of the problem being solved.*")

Related on Giddings' classification the software in our system belongs to the category of embedded domain dependent software, which represents the highest complexity.

The development of the system required accuracy and obeying the principles that we suggest as general principles for development of virus execution systems. In Chapter 2 we presented the following principles:

- 1) The system must be isolated in such a way that a possibly escaped virus cannot cause harm to external computer systems.
- 2) The system must be designed as much as possible in such a way that a malicious code executed in a controlled environment cannot harm the system.
- 3) The system should be designed to be flexible in order to allow flexible future development.
- 4) The system should be designed to work as continuously as possible

The first condition was met by isolating the system from external connections and by verifying the integrity of executable files of the system. This was a successful method to prevent viruses from escaping. One could argue that isolation causes such inconvenience that it is not profitable. However, we decided to put safety ahead of convenience. Our argument is that safety violations can cause so many difficulties that safety is essential condition for computer antivirus research work. An antivirus organisation seen to be neglecting safety can be withdrawn from co-operation with other antivirus organisations.

The second condition was met by carefully preparing for possible vulnerabilities of the system. This included carefully examining all possible ways how a virus could compromise the system. Therefore, as an instance, flash BIOS was write-protected in order to prevent malicious programs from writing over information of the flash BIOS, the Victim PC had restricted access to the network even in clean stage and care was taken that the Victim PC could boot from the network only when the Monitoring PC had authorised the operation.

The third requirement was fulfilled by designing the software components of the system as flexible as possible. As the recovery could be accomplished from a bit to bit image file, the Victim PC's operating system did not matter. Furthermore, as the keyboard controlling was not dependent on the operating environment of the Victim PC, it could be applied to various operating environments. However, we must note the exception that when going from DOS to Windows, the keyboard-controlling device no longer worked and had to be realised again. The timing program running on the Monitoring PC was designed to be able to call any other programs whenever necessary. This allowed the same monitoring program to work flexibly for different operations.

The fourth condition was met by solving such problems that would have precluded the system from working continuously. This included such as implementing automatic cold boot in such a way that the boot operation occurred no matter in which stage the Victim PC was. Furthermore, CMOS memory failures were solved and hard disk was automatically low-level formatted, if necessary.

7.2 Limitations

Despite all the advantages of this dissertation there are also limitations. We will continue by discussing the limitations of the results from Chapters 3, 4, 5 and 6.

In Chapter 3 we discussed the terminology associated with computer viruses and malicious program code. We must remember that some of the definitions are still argued and there exist different definitions. Furthermore, in some cases there can be so-called grey areas, where it is difficult or even impossible to decide unequivocally in which category the program code should belong. However, we must also remember that this is a general problem concerning program code classification. Concerning the classifications discovered in the Chapter 3 we must remember that there can also exist other ways to classify malicious program code. As an instance, if we take the characteristics of the viruses presented in Subsection 3.4, the classification depends on which characteristics are included in the classification.

In Chapter 4 we constructed the theoretical framework for computer antivirus product virus detection analysis. There may be areas of virus detection analysis that have not been discussed. Virus detection analysis is based on viruses and therefore detection analysis of other malicious program code has not been discussed although I recognise the importance of malware prevention.

In Chapter 5 we discussed the construction of computer-supported processes. Despite all the advantages of computer-supported virus replication processes there are also disadvantages. One drawback is that not all possible viruses can be replicated, because some viruses may spread only under special conditions, which the system does not completely identify. We can conclude that a sample file can be proved to contain a virus when infection occurs, but the opposite cannot be proved, if infection does not occur. Manual virus analysis may be required for analysis of the replication mechanism. Nevertheless, the system saves enormous work effort, because most of the replication mechanisms can be easily covered.

Macro viruses are also concerned with the same drawback that not all possible viruses can be replicated, because some viruses may spread only under some special conditions, which the system does not completely identify. Macro viruses can use countless different ways for infecting documents (Bontchev, 1996) and it is not possible to cover all of them. The system can be built to sense macros in documents, but still not all different infection mechanisms can be covered.

Furthermore, as Tocheva (2001) demonstrates, self-distributing viruses can use various replication mechanisms. Covering all of them with a simple system setting is not possible. In fact, in the case of self-distributing viruses the replication mechanism should be known in advance so that the system configuration will match the replication mechanism. The requirement for different system configurations will also raise the problem that there is a need to construct several different system settings.

Antivirus products typically have two operations, which are virus detection and virus disinfection. Although I consider disinfection an important feature of computer antivirus products, we have not presented how disinfection capabilities could be evaluated. The reason for this is that so far the system has not been utilised for this particular purpose. As Bontchev has demonstrated (1998, pp. 242-244) antivirus products' disinfection capabilities are more difficult to evaluate than virus detection capabilities and special evaluation methods are needed.

In Chapter 6 we discussed the efficiency of computer-supported processes compared to manual processes. Although the chapter concentrated on efficiency, there may be also other methods for assessing the processes. For example, there exist such properties as experiential knowledge gathered, quality of the processes, construction efforts and applicability of the processes. One limitation is that we did not discuss such qualities in detail.

The results received from manual virus replication processes can vary depending on the software tools, personal capabilities, computer systems, sample files and casualty. Therefore the results received from manual replication are only rough estimates of the real used time. Furthermore, results gained from automatic replication processes can vary depending on sample files used, replication system used and configurations. We can conclude that the assessment of the processes cannot give exact results, but rather rough estimates of the time used. Nevertheless, the results received show the difference between automatic and manual methods. Furthermore, we must remember that automated methods free one person to do other tasks whereas manual processes occupy at least one person.

One drawback of the system is that it has not been built to operate as quickly as possible, because the emphasis has been on reliability. Therefore the results from the self-assessment of the system are not as optimal as they could be.

Furthermore, we did not count system halts, which are likely to occur to some extent in a real situation. Therefore the results may not reflect a real world situation. Pauses in system usage, full network server disk space, errors in software or hardware dysfunctions can cause system halts. However, our experiment suggests that the system can be built to work reliably and possible system dysfunctions can be quickly fixed. Furthermore, our intention was to estimate optimum performance of the system and therefore it would not be meaningful to count natural non-usage of the system.

While assessing the system performance in Subsection 6.3 it is possible that we have made few errors while counting processing time. If the replication processes stopped for some reason, we have excluded such cases, because our intention is to estimate optimum processing time. However, the end of processing time was not recorded, if the system was halted. Furthermore, there was no indication whether processing stopped because of normal operation or dysfunction of the system. Normal operations include cases when the sample files put for replication ran out, the server ran out of disk space or the processing was stopped.

The manual replication process was constructed with the same methods as the automatic replication process. In a real situation with manual macro virus replication there are probably more optimised and thus less time consuming processes in use than we have used. The optimisation could be done by recovering only changed system areas and by writing a quicker macro checker. However, the same optimisation could also have been realised with automatic macro virus replication. The reason why this is not realised at the moment of finishing writing this thesis is a matter of emphasis. During the system development we decided to emphasise more reliability and flexibility than time consumption.

7.3 Recommendations to practitioners

The results achieved in Chapter 3 can be used as a general guideline to classify malicious program code. The results can be used as a basis for naming schemes. We recognise that there can be different classifying methods, but as a general guideline the classification schemes should be as exact as possible in order to exclude ambiguous cases.

The results achieved in Chapter 4 can be used as a general guideline for antivirus product virus detection analysis. The findings discussed in the chapter need to be realised when analysing antivirus products' capabilities to detect viruses.

In Chapter 5 we showed that many processes associated with computer antivirus product virus detection analysis can be automated. Such processes can also be applied to automating tasks in antivirus product quality control. If the processes were systematically applied, product quality would evidently improve as flaws could be found efficiently. The principles of the Automatic and Controlled Virus Code Execution System could also be adapted to other applications than those presented in this thesis. Unfortunately, we cannot foresee all of these, but these tasks could include such tasks as facilitating antivirus research, facilitating computer security research and automated analysis of user interfaces.

We believe that the difference between manual and automatic replication processes is so obvious that we can conclude that automatic processes are more efficient than manual processes when the need for processing is continuous. Therefore our conclusion in Chapter 6 is that the development of an automated system is cost-effective, if the need for processing is continuous.

7.4 Recommendations to researchers

During our study, especially writing Chapters 1,3,4,5 and 6 we found some new research areas. We will next suggest possible future steps.

As discovered in Chapter 1, computer ethics is one important aspect of computer antivirus research. Therefore research in this area is needed for establishing guidelines, solving dilemmas and finding perspectives.

As demonstrated in Chapter 3, the classification of malicious software is one difficult dilemma. It is sometimes difficult to decide in which category a program code belongs. Brunnstein has approached this problem (1999) from the aspect of software dysfunctions. However, further research on malware classification is needed. One important object in the research of this area is to standardise concepts in such a way that ambiguity can be solved.

The theoretical classification of computer antivirus product's virus detection analysis discussed in Chapter 4 does not include detailed discussion of virus attack emulation and vulnerability analysis methods. These are important future research areas, because these methods allow successful analysis of nonidentifying antivirus products' virus detection capabilities.

Furthermore, we concluded in Chapter 4 that there cannot be an exact estimation for an antivirus product's sensitivity to false positives. Nevertheless, the analysis of antivirus product's sensitivity to false alarms is one important area to research. Moreover, we found important research areas in developing metrics for measuring commonness and replication capabilities of viruses.

We concentrated on virus detection analysis and therefore we delimited other areas of antivirus product evaluation from this dissertation. Important research areas can be found, for example, from assessing technical support and usability. Furthermore, since self-distributing viruses may replicate quickly around the world it is important to develop metrics measuring antivirus product's capabilities prevent self-distributing viruses.

The Virus Test Center has begun using other malware than viruses in antivirus product analyses (1999). Furthermore, I have studied the possibilities of using malware in antivirus product evaluation (Helenius 1999b). However, because of the intricacy of malware testing future research in the area of malware classification, prevention and detection analysis is needed.

One research area that I observed during antivirus product evaluation is the need to exactly identify viruses. Therefore the valuable virus naming and classification work Virus Test Center has put into practise should be continued, developed and followed.

The Automatic and Controlled Virus Code Execution System discussed in Chapter 5 has been designed to be flexible and from this it follows that the system has been designed for future needs. The principles of the system could also be applied for other applications than those presented in this thesis. I have so far only described such applications for which the system has been used, but there are also other tasks which can be automated by utilising the system. In general, this includes all such tasks which require systematic automation and which can be automated by controlling a keyboard and boot device selection. Moreover, as the system was designed to be flexible there can be other applications for the system that cannot yet be foreseen.

One research area is to construct processes estimating how well antivirus products can prevent viruses from spreading in different ways. For example, documents can be opened in different ways, or a computer could be infected by a virus before an antivirus product is used. The Automatic and Controlled Virus Code Execution System could be applied to estimate how well antivirus products can prevent viruses coming via the Internet. This includes such as preventing viruses from coming via e-mail attachments, ftp or World Wide Web.

One future direction is to improve the Automatic and Controlled Virus Code Execution System. One obvious enhancement is to improve the self-e-mail replication process. Furthermore, viruses using certain Internet addresses could be simulated. The self-e-mail replication process can easily be extended to different operating systems and e-mail programs. The drawback is that several configurations may be needed in order to successfully replicate viruses. The enhancement does not need to concern only self-e-mailing viruses but also other types of self-distributing virus. The enhanced self-e-mail replication process could handle, for example, viruses using WWW-pages and vulnerabilities (see for example descriptions of the Code Red and Nimda viruses in CERT 2001b and CERT 2001c).

As stated, optimum speed has not been a major goal of the system. However, as there is a need for a growing number of different configurations with growing complexity of operating systems and applications, there is a need for optimisation. Infection analysis could be optimised by enhancing our internal macro checker and recovery could be optimised by recovering only such system areas that have been changed.

Although some antivirus product evaluators have evaluated disinfection capabilities, this has mainly been haphazard and disinfection analysis methods are therefore one important area to research. Assessment of antivirus products' disinfection capabilities could be at least partly automated by the automated system. The results in Chapter 6 concentrated on the efficiency of the system compared to manual processes. However, there are also other aspects that could by evaluated. For both manual and automated processes such factors as likelihood for errors and successfulness of replication processes could be studied. Furthermore, efficiency of tasks associated with self-distributing viruses could be evaluated.

It seems that an assessment of a high complexity virus code execution system requires some tailored metrics compared to those applicable to traditional software engineering. One research area is to develop metrics applicable to virus code execution systems. Our proposal for the most important metrics required for assessing virus code execution systems are the following.

- Metrics measuring functionality of the system. The functions of the Automatic and Controlled Virus Code Execution System were discussed in Chapter 5.
- Metrics measuring efficiency of the system. The efficiency of the Automatic and Controlled Virus Code Execution System is discussed in Chapter 6. Related to efficiency such characteristics can be assessed as how continuously the system can be used. For assessing continuity the system should have built-in capabilities to keep track of dysfunctions. We have not presented data for continuity, but in our experience the continuity of the Automatic and Controlled Virus Code Execution System is near to optimum in virus replication processes. This means a high probability that the system is able to complete a once initiated process.
- Metrics measuring output of the system. The system output is the result of a process. Depending on the process, the output can include a virus replicated to new objects, data from the process and so on.
- Metrics measuring flexibility of the system. Although it is difficult to find metrics for flexibility, flexibility allows a system to be adapted for future needs. The flexibility of the Automatic and Controlled Virus Code Execution System was demonstrated in Chapter 5.

We have now discussed the importance of the results, limitations, and suggested possible future steps. As we can see, the construction of computersupported processes has resulted in interesting possibilities and future development areas.

References

Aubrey-Jones David (1995), "Automatic Testing of Memory Resident Scanners", In the proceedings of the Fifth International Virus Bulletin Conference held in Boston, U.S.A. September 1995, hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 125-132

AVIEN (2001), "Anti-Virus Information Exchange Network Code of Conduct", Available: http://www.avien.org/codeconduct.html (14.3.2002)

Bechtel Kenneth (2001), "Tilting at Windmills", Virus Bulletin Journal, December 2001, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 9-10, Subscriptions available at http://www.virusbtn.com (22.2.2001)

Bontchev Vesselin (1992), "Possible Virus Attacks Against Integrity Programs and How to Prevent Them". Proceedings in the 2nd International Virus Bulletin Conference. September 1992. pp. 131-141. Available: http://www.complex.is/~bontchev/ (27.7.2001)

Bontchev Vesselin (1993), "Analysis and Maintenance of a Clean Virus Library", Proceedings of 3rd International Virus Bulletin Conference, pp. 77-89 Available http://www.complex.is/~bontchev/ (27.7.2001)

Bontchev Vesselin (1996), "Possible Macro Virus Attacks and How to Prevent Them", In proceedings of the International EICAR Conference 1996, Lintz, Austria. Hosted by DataPROT Linz. pp. 61-87.

Bontchev Vesselin (1998), "Methodology of Anti-Virus Research", Dissertation, Faculty of Informatics, University of Hamburg, 1988

Bontchev Vesselin (1999), "The Wildlist - Still Usefull", Proceedings of 9th International Virus Bulletin Conference, pp. 281-287, Available: http://www.complex.is/~bontchev/ (27.7.2001)

Brunnstein Klaus (1999), "From AntiVirus to AntiMalware Software and Beyond: Another Approach to the Protection of Customers from Dysfunctional System Behaviour." In proceedings of 22nd National Information Systems Security Conference Available: http://csrc.nist.gov/nissc/1999/proceeding/papers/p12.pdf (25.7.2001)

Debian (2001), "What is Debian?", http://www.debian.org/ (25.7.2001)

CERT Coordinator Center (2001a), "Vulnerabilities, Incidents & Fixes", Available: http://www.cert.org/nav/ (28.9.2001)

CERT Coordinator Center (2001b), "CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL", Available: http://www.cert.org/advisories/CA-2001-19.html (11.1.2002)

CERT Coordinator Center (2001c), "CERT Advisory CA-2001-26 Nimda Worm" Available: http://www.cert.org/advisories/CA-2001-26.html (11.1.2002)

Chistopher Klaus (1996), "Security-Patches FAQ", Available: http://www.faqs.org/faqs/computer-security/security-patches (28.9.2001)

Coursen Shane (1996), "How Much is that Virus in the Window?", Virus Bulletin Journal, December 1996, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 13-17, Subscriptions available at http://www.virusbtn.com (22.2.2001)

Cohen Fred (1986), "Computer Viruses", Dissertation, Faculty of the Graduate School, University of Southern California

EICAR (1999a), "Our mission", http://www.eicar.org/mission.htm, (25.7.2001)

EICAR (1999b), "Code of Conduct", http://www.eicar.org/code_of_conduct.htm, (25.7.2001)

Floyd Christiane, Reisin Fanny-Michaela and Schmidt Gerhard (1989); "STEPS to Software Development with Users"; In proceedings of 2nd European Software Engineering Conference (ESEC) University of Varwick, Covently, UK 11-15.9 1989

Giddings Richards (1984), "Accommodating Uncertainty in Software Design", Communications of the ACM, Vol. 27 No 5, May 1984, pp. 428-434

Gryaznov Dmitry (1994), "Simboot: A New Tool for Testing Scanners", In proceedings of the EICAR 1994 conference held in London, England 23.-25.11 1994, hosted by S&S International Plc. pp. 157-164

Helenius Marko (1994a), "Antivirus Scanner Analysis by Using the "In the Wild" Test Set", 18.11.1994, Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/wildtest.zip (29.1.1998)

Helenius Marko (1994b), "Tietokonevirukset ja virustentorjunta", 1994, University of Tampere, Department of Computer Science, Report B, B-1994-3, Tampere University press

Helenius Marko (1995a), "Automatic and Controlled Virus Code Execution System", In proceedings of the EICAR 1995 Conference held in Zürich, Switzerland 27.-29.11 1995, hosted by CIMA AG. pp. T3 13-21. Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/automat.zip (29.1.1998)

Helenius Marko (1995b), "Antivirus Scanner Analysis 1995", Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/test1995.zip (29.1.1998)

Helenius Marko (1996a), "Problems with Analysing Computer Antivirus Software and Some Possible Solutions", In proceedings of the International EICAR Conference 1996, Lintz, Austria, Hosted by DataPROT Linz. pp. 96-102 (29.1.1998)

Helenius Marko (1996b), "Antivirus Scanner Analysis Based on Joe Well's List of PC Viruses in the Wild 3/1996", Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/test1996.zip (29.1.1998)

Helenius Marko (1997), "Antivirus Scanner Analysis Based on Joe Well's List of PC Viruses in the Wild 7/1997", Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/test1997.zip (29.2.1998)

Helenius Marko (1998a), "Automatic and Controlled Macro Virus Execution and Automating the Windows Environment", In proceedings of the EICAR 1998 Conference held in Munich, Germany 16.-18.3 1998, Available Available via anonymous ftp as: ftp://ftp.cs.uta.fi/pub/vru/documents/autowin.zip (2.8.1998)

Helenius Marko (1998b), "Automating Antivirus Product Evaluation", In proceedings of the Virus Bulletin 1998 Conference held in Munich, Germany 22-23.10 1998, Subscriptions available at http://www.virusbtn.com, pp. 251-260

Helenius Marko (1999a), "Antivirus Scanner Analysis Based on Joe Well's List of PC Viruses in the Wild 10/1998", Available via anonymous ftp as ftp://ftp.cs.uta.fi/pub/vru/documents/test1999.zip (22.8.2001)

Helenius Marko (1999b). "Problems, Advantages and Disadvantages of Malware Testing". EICAR Prodeedings 1999 Editors: Gattiker Urs, Pedersen Pia ja Petersen Karsten. EICAR c/o TIM-World ApS. Aalborg. Denmark 1999. Available on "EICAR-99 Conference Proceedings" CD-ROM. Subscriptions available at http://www.eicar.org

Järvinen Pertti (1999), "On Research Methods", Opinpaja Oy, Tampere, Finland, 1999

Kephart Jeffrey and Arnold William (1994), "Automatic Extraction of Computer Virus Signatures" In Proceedings of the 4th International Virus Bulletin Conference, Editor Ford Richard., Virus Bulletin Ltd., Abingdon, England, pp. 178-184 Available: http://www.research.ibm.com/antivirus/SciPapers/Kephart/VB94/vb94.html (27.7.2001)

Kephart Jeffrey, Sorkin Gregory, Swimmer Morton and White Steve (1997), "Blueprint for a Computer Immune System" In Proceedings of the 7th International Virus Bulletin Conference, Virus Bulletin Ltd., Abingdon, England, Available: http://www.research.ibm.com/antivirus/SciPapers/Kephart/VB97/ (31.7. 2001)

Leitold Ferench (1995), "Automatic Virus Analyser System", In the proceedings of the Fifth International Virus Bulletin Conference held in Boston, U.S.A. September 1995. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 99-108

Mason Richard (1988); "Experimentation and Knowledge"; Knowledge: Creation, Diffusion, Utilization; Vol. 10 No. 1, September 1988, pp. 3-24

Muttik Igor (1995), "The Problem in Creating Goat Files", In the proceedings of the Fifth International Virus Bulletin Conference held in Boston, USA September 1995. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 109-124

Muttik Igor (2000), "Stripping Down an AV Engine", In the proceedings of the Tenth International Virus Bulletin Conference held in Boston, USA September 2000. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 59-68. Available: http://www.virusbtn.com/vb2000/Programme/ (27.7.2001)

Pressman Roger (2001), "Software Engineering - A Practitioner's Approach", European Adaptation, Fifth Edition, Boston, McGraw-Hill Companies Inc.

Salvatore March, Gerald Smith (1995), "Design and Natural Science on Information Technology", Decision Support Systems 15, Elsevier Science Publishers Ltd., pp. 251-266

Skulason Fredrik (1994), "The Virus Glut - The Impact of the Virus Flood", In the proceedings of the EICAR 1994 conference held in London, England 23.-25.11 1994, hosted by S&S International Plc. pp. 143-147

Solomon Alan and Kay Tim (1994), "Dr Solomon's PC Anti-virus Book", Newtech. Oxford 1994. pp. 13-18

Stojakovic-Celustka Suzana (2000), "Building Secure Information Systems", Dissertation, Czech Technical University in Prague, Department of Computer Science & Engineering, Available: http://www.geocities.com/suzana_sc2001/index.htm (11.1.2002)

Swimmer Morton (1995), "Virus Intrusion Detection Expert System", In the proceedings of the EICAR 1995 Conference held in Zürich, Switzerland 27.-29.11 1995.

Ször Peter (2000), "Attacks on Win32 - Part II", In the proceedings of the Tenth International Virus Bulletin Conference held in Boston, USA September 2000. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 47-68. Available: http://www.virusbtn.com/vb2000/Programme/ (27.7.2001)

Tocheva Katrin (2001). "Worming the Internet", Parts 1, 2 and 3. Virus Bulletin Journal. October, November & December 2001. Virus Bulletin Ltd. 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS. England. pp. 10-11, 12-13 and 13-15. Subscriptions available at http://www.virusbtn.com (22.2.2001)

Trend Micro (2001), "Trend World Virus Tracking Center", http://www.trendmicro.com/wtc/ (26.11.2001)

Trendy Christine (1996), "Wacky Widgets, Wacky Costs: False Positives", Virus Bulletin Journal, May 1996, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 16-17. Subscriptions available at http://www.virusbtn.com (22.2.2001)

Veldman Frans (1995), "Why Do We Need Heuristics?", In the proceedings of the Fifth International Virus Bulletin Conference held in Boston, USA September 1995. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. XI-XV

Virus Bulletin (1995), "Virus Total Reaches 5000", Virus Bulletin Journal, October 1994, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 3. Subscriptions available at http://www.virusbtn.com (22.2.2001)

Virus Bulletin (1995), "Editorial - When Techniques Jump Fences", Virus Bulletin Journal, September 1995, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 2. Subscriptions available at http://www.virusbtn.com (22.2.2001)

Virus Bulletin (1995-1998), "Virus Prevalence Tables", Virus Bulletin Journal, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. Subscriptions available at http://www.virusbtn.com (22.2.2001)

Virus Test Center (1994-2002), "Anti-Virus/AntiMalware Scanner-Tests" available: http://agn-www.informatik.uni-hamburg.de/vtc/naveng.htm (10.1.2002)

Virus Test Center (2001), "Scanner test October 2001", section "Prologue of VTC test", available: http://agn-www.informatik.uni-hamburg.de/vtc/en0110.htm (10.1.2002)

Whalley Ian (2000), "Controlled Worm Replication - "Internet in a box"" In the proceedings of the Tenth International Virus Bulletin Conference held in Boston, USA September 2000. hosted by Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 43-65. Available: http://www.virusbtn.com/vb2000/Programme/ (27. 7.2001)

The WildList Organization International (1993-2002), "PC Viruses in the Wild", http://www.wildlist.org/WildList/ (6.8.2001)

Zwienenberg Righard (2001), "Feature 1 - Scriptography", Virus Bulletin Journal, July 2001, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England. pp. 10-11. Subscriptions available at http://www.virusbtn.com (22.2.2001)

APPENDIX 1: DEFINITIONS OF SOME TERMS

antivirus product evaluation: an assessment of computer antivirus products' properties. Typically antivirus product evaluation compares the capabilities of different antivirus products. Virus detection analysis is one part of antivirus product evaluation.

antivirus product virus detection analysis: an analysis which estimates antivirus products' virus detection capabilities

CMOS memory: CMOS (complementary metal oxide semiconductor) memory is a low powered electrical memory. In computer systems CMOS memory is typically battery powered and contains such system settings that remain in memory when the main power is off. This includes such as system date, system time, hard disk settings, floppy diskette drive settings, read access memory settings and boot order.

CMOS memory failure: a situation where a computer's CMOS memory's content has changed abnormally. Execution of some malicious program code can cause CMOS memory failures.

cold boot: computer system boot done in such a way that the main electricity of the computer is physically switched first off and then back on.

false alarm: a situation in which an antivirus product announces that it has found a virus, when in reality there is no virus on the object in question.

goat file: a file that is created to be infected by a virus. Typically a goat file is written in such a way that it facilitates virus disassembly and virus infection can be easily observed.

image file: a file consisting of a bit to bit copy of a data storage medium. Typically an image file is written from a hard disk or a floppy diskette.

virus detection analysis A method to analyse computer antivirus product's capabilities to detect viruses.

virus test bed: a specially prepared set of virus samples meant to be used for computer antivirus product evaluation. Typically a virus test bed is prepared so that there are several specimens per each virus and an important objective in preparing a test bed is to ensure that each virus specimen is capable of replicating recursively.

vulnerability analysis: an analysis that investigates an antivirus product's capability to prevent or detect different types of attack typical for viruses.