Computer Viruses as Artificial Life*

Eugene H. Spafford
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
spaf@cs.purdue.edu

Abstract

There has been considerable interest in computer viruses since they first appeared in 1981, and especially in the past few years as they have reached epidemic numbers in many personal computer environments. Viruses have been written about as a security problem, as a social problem, and as a possible means of performing useful tasks in a distributed computing environment.

However, only recently have some scientists begun to ask if computer viruses are not a form of artificial life — a self-replicating organism. Simply because computer viruses do not exist as organic molecules may not be sufficient reason to dismiss the classification of this form of "vandalware" as a form of life.

This paper begins with a description of how computer viruses operate and their history, and of the various ways computer viruses are structured. It then examines how viruses meet properties associated with life as defined by some researchers in the area of artificial life and self-organizing systems. The paper concludes with some comments directed towards the definition of artificially "alive" systems and experimentation.

1 Introduction

There has been considerable interest in computer viruses during the last several years. One aspect of this interest has been to ask if computer viruses are a form of artificial life, and what that might imply. To address this, we must first understand something of the history and structure of computer

Portions of this article are derived from [31], [30] and [28].

* To appear in the *Journal of Artificial Life*, MIT Press, 1994.

viruses. Thus, we will begin with a condensed, high-level description of computer viruses — their history, structure, and how they relate to some properties that might define artificial life.

A more detailed introduction to the topic of computer viruses may be found in the references, particularly [9, 2, 3, 5] and [15]. Also of use are [11, 14, 10, 16] and [24], although the lists presented in the latter are somewhat out of date.

2 What is a Computer Virus?

Computers are designed to execute instructions one after another. Those instructions usually do something useful — calculate values, maintain databases, and communicate with users and with other systems. Sometimes, however, the instructions executed can be damaging and malicious in nature. When that happens by accident, we call the code involved a software *bug*[1] — perhaps the most common cause of unexpected program behavior. If the source of the instructions was an individual who intended that the abnormal behavior occur, then we consider this malicious coding; authorities have sometimes referred to this code as *malware* and *vandalware*. These names relate to the usual effect of such software.

There are many distinct forms of this software that are characterized by the way they behave, how they are triggered, and how they spread. In recent years, occurrences of malware have been described almost uniformly by the media as *computer viruses*. In some environments, people have been quick to report almost every problem as the result of a virus. This is unfortunate, as most problems are from other causes (including, most often, operator error). Viruses are widespread, but they are not responsible for many of the problems attributed to them.

The term *computer virus* is derived from and is in some sense analogous to a biological virus. The word *virus* itself is Latin for *poison*. Simplistically, biological viral infections are spread by the virus (a small shell containing genetic material) injecting its contents into a far larger organism's cell. The cell then is infected and converted into a biological factory producing replicants of the virus.

Similarly, a computer virus is a segment of machine code (typically 200-4000 bytes) that will copy itself (or a modified version of itself) into one or more larger "host" programs when it is activated. When these infected programs are run, the viral code is executed and the virus spreads further. Sometimes, what constitutes "programs" is more than simply applications: boot code, device drivers, and command interpreters also can be infected.

Computer viruses cannot spread by infecting pure data; pure data files are not executed. However, some data, such as files with spreadsheet input or text files for editing, may be interpreted

[1] The original choice of the term "bug" is unfortunate in this context, and is unrelated to the topic of artificial life.

by application programs. For instance, text files may contain special sequences of characters that are executed as editor commands when the file is first read into the editor. Under these circumstances, the data files are "executed" and may spread a virus. Data files may also contain "hidden" code that is executed when the file is used by an application, and this too may be infected. Technically speaking, however, pure data itself cannot be infected by a computer virus.

The first use of the term *virus* to refer to unwanted computer code was by the science fiction author David Gerrold. He wrote a series of short stories about a fictional G.O.D. machine (super computer) in the early 1970s that were later merged into a novel in 1972: *When Harlie Was One*.[12] The description of *virus* in that book does not fit the currently-accepted, popular definition of computer virus — a program that alters other programs to include a copy of itself.

Fred Cohen formally defined the term *computer virus* in 1983.[2] At that time, Cohen was a graduate student at the University of Southern California attending a security seminar. Something discussed in class inspired him to think about self-reproducing code. He put together a simple example that he demonstrated to the class. His advisor, Professor Len Adleman, thinking about the behavior of this creation, suggested that Cohen call his creation a computer virus. Dr. Cohen's Ph.D. thesis and later research were devoted to computer viruses.

Actual computer viruses were being written by individuals before Cohen, although not named such, as early as 1980 on Apple II computers.[9] The first few viruses were not circulated outside of a small population, with the notable exception of the "Elk Cloner" virus released in 1981 on several bulletin board systems.

Although Cohen (and others, including Len Adleman[1]) have attempted formal definitions of *computer virus*, none have gained widespread acceptance or use. This is a result of the difficulty in defining precisely the characteristics of what a virus is and is not. Cohen's formal definition includes any programs capable of self-reproduction. Thus, by his definition, programs such as compilers and editors would be classed as "viruses." This also has led to confusion when Cohen (and others) have referred to "good viruses" — something that most others involved in the field believe to be an oxymoron.[4, 29]

Stubbs and Hoffman quote a definition by John Inglis that captures the generally accepted view of computer viruses:

> "He defines a virus as a piece of code with two characteristics:
> 1. At least a partially automated capability to reproduce.
> 2. A method of transfer which is dependent on its ability to attach itself to other computer entities (programs, disk sectors, data files, etc.) that move between these systems."[32, p. 145]

Several other interesting definitions are discussed in [14, Chapter 1].

After first appearing as a novelty, true computer viruses have become a significant problem. In particular, they have flourished in the weaker security environment of the personal computer. Personal computers were originally designed for a single dedicated user — little, if any, thought was given to the difficulties that might arise should others have even indirect access to the machine. The systems contained no security facilities beyond an optional key switch, and there was a minimal amount of security-related software available to safeguard data. Today, however, personal computers are being used for tasks far different from those originally envisioned, including managing company databases and participating in networks of computer systems. Unfortunately, their hardware and operating systems are still based on the assumption of single trusted user access, and this allows computer viruses to spread and flourish on those machines. The population of users of PCs further adds to the problem, as many are unsophisticated and unaware of the potential problems involved with lax security and uncontrolled sharing of media.

Over time, the problem of viruses has grown to significant proportions. In the seven years after the first infection by the *Brain* virus in January 1986, generally accepted as the first significant MS-DOS virus, the number of known viruses has grown to several thousand different viruses, most of which are for MS-DOS.

The problem has not been restricted to the IBM PC, however, and now affects all popular personal computers. Mainframe viruses may be written for any operating system that supports sharing of data and executable software, but all reported to date have been experimental in nature, written by serious academic researchers in controlled environments (e.g., [6]). This is probably a result, in part, of the greater restrictions built into the software and hardware of those machines, and of the way they are usually used. It may also be a reflection on the more technical nature of the user population of these machines.

2.1 Related Software

Worms are another form of software that is often referred to as a computer virus. Unlike viruses, worms are programs that can run independently and travel from machine to machine across network connections; worms may have portions of themselves running on many different machines. Worms do not change other programs, although they may carry other code that does, such as a true virus. It is their replication behavior that leads some people to believe that worms are a form of virus, especially those people using Cohen's formal definition (which incidentally would also classify standard network file transfer programs as viruses). The fact that worms do not modify existing programs is a clear distinction between viruses and worms, however.

In 1982, John Shoch and Jon Hupp of Xerox PARC (Palo Alto Research Center) described the first computer worms.[23] They were working with an experimental, networked environment using one of the first local area networks. While searching for something that would use their networked environment, one of them remembered reading *The Shockwave Rider* by John Brunner, written in 1975. This science fiction novel described programs that traversed networks, carrying information with them. Those programs were called *tapeworms* in the novel. Shoch and Hupp named their own programs *worms*, because they saw a parallel to Brunner's tapeworms. The Xerox worms were actually useful — they would travel from workstation to workstation, reclaiming file space, shutting off idle workstations, delivering mail, and doing other useful tasks.

The Internet Worm of November 1988 is often cited as the canonical example of a damaging worm program.[26, 27, 22] The Worm clogged machines and networks as it spread out of control, replicating on thousands of machines around the Internet. Some authors (e.g., [7]) labeled the Internet Worm as a virus, but those arguments are not convincing (cf. the discussion in [25]). Most people working with self-replicating code now accept the Worm as a form of software distinct from computer viruses.

Few computer worms have been written in the time since then, especially worms that have caused damage, because they are not easy to write. Worms require a network environment and an author who is familiar not only with the network services and facilities, but also with the operating facilities required to support them once they have reached their targets.

Worms have also appeared in other science fiction literature. Recent "cyberpunk" novels such as *Neuromancer* by William Gibson [13] refer to worms by the term "virus." The media has also often referred incorrectly to worms as viruses. This paper focuses only on viruses as defined above. Many of the comments about viruses and artificial life may also be applied to worm programs.

Harold Thimbleby coined the term *liveware* to describe another form of self-propagating software that carries information or program updates.[33] Liveware shares many of the characteristics of both viruses and worms, but has the additional distinction of announcing its presence and requesting permission from the user to execute its intended functions. There have been no reports of liveware being discovered or developed other than by Thimbleby and his colleagues.

Other forms of self-reproducing and usually malicious software have also been written. Although no formal definitions have been accepted by the entire community to describe this software, there are some informal definitions that seem to be commonly accepted (cf. [21]). Several of these are often discussed by analogy to living organisms. This tendency towards anthropomorphism has perhaps led to some confusion about the nature of this software. Rather than discuss each of these software forms here, possibly adding to the confusion, the remainder of this paper will focus on computer viruses only; the interested reader may peruse the cited references.

3 Virus Structure and Operation

True viruses have two major components: one that handles the spread of the virus, and a "payload" or "manipulation" task. The payload task may not be present (has null effect), or it may await a set of predetermined circumstances before triggering.

For a computer virus to work, it somehow must add itself to other executable code. The viral code is usually executed before the code of its infected host (if the host code is ever executed again). One form of classification of computer viruses is based on the three ways a virus may add itself to host code: as a shell, as an add-on, and as intrusive code.

A fourth form, the so-called *companion virus*, is not really a virus at all, but a form of *trojan horse* that uses the execution path mechanism to execute in place of a normal program. Unlike all other viral forms, it does not alter any existing code in any fashion: companion viruses create new executable files with a name similar to an existing program, and chosen so that they are normally executed prior to the "real" program. As companion viruses are not real viruses unless one uses a more encompassing definition of virus, they will not be described further here.

Shell viruses A shell virus is one that forms a "shell" (as in "eggshell" rather than "Unix shell") around the original code. In effect, the virus becomes the program, and the original host program becomes an internal subroutine of the viral code. An extreme example of this would be a case where the virus moves the original code to a new location and takes on its identity. When the virus is finished executing, it retrieves the host program code and begins its execution.

Almost all boot program viruses (described below) are shell viruses.

Add-on viruses Most viruses are add-on viruses. They function by appending their code to the host code, and/or by relocating the host code and inserting their own code to the beginning. The add-on virus then alters the startup information of the program, executing the viral code before the code for the main program. The host code is left almost completely untouched; the only visible indication that a virus is present is that the file grows larger, if that can indeed be noticed.

Intrusive viruses Intrusive viruses operate by overwriting some or all of the original host code with viral code. The replacement might be selective, as in replacing a subroutine with the virus, or inserting a new interrupt vector and routine. The replacement may also be extensive, as when large portions of the host program are completely replaced by the viral code. In the latter case, the original program can no longer function properly. Few viruses are intrusive viruses.

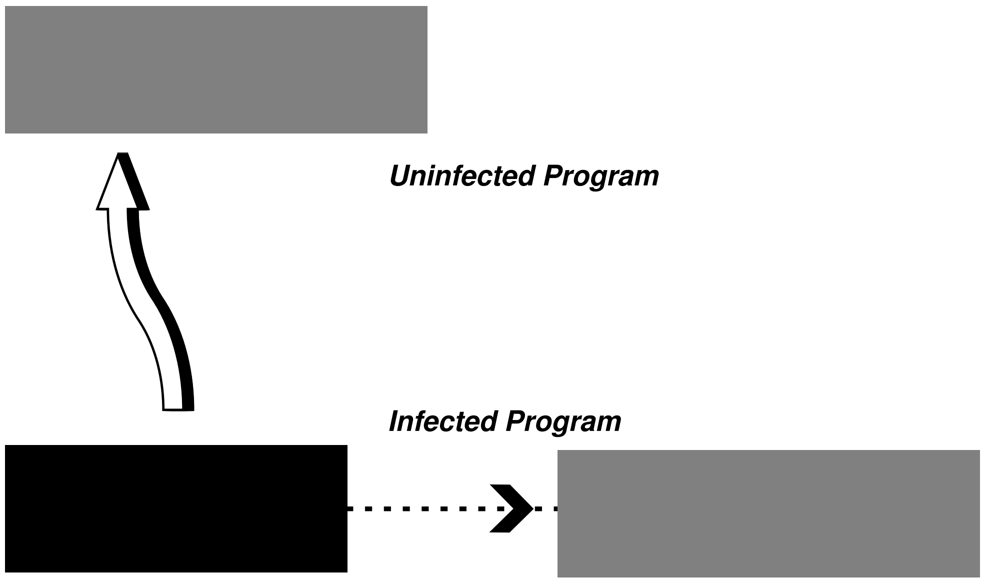

Figure 1: Shell Virus Infection

A second form of classification used by some authors (e.g., [24]) is to divide viruses into file infectors and boot (system startup) program infectors. This is not particularly clear, however, as there are viruses that spread by altering system-related code that is neither boot code nor programs. Some viruses target file system directories, for example. Other viruses infect both application files *and* boot sectors. This second form of classification is also highly specific and only makes sense for machines that have infectable (writable) boot code.

Yet a third form of classification is related to how viruses are activated and select new targets for alteration. The simplest viruses are those that run when their "host" program is run, select a target program to modify, and then transfer control to the host. These viruses are *transient* or *direct* viruses, known as such because they operate only for a short time, and they go directly to disk to seek out programs to infect.

The most "successful" PC viruses to date exploit a variety of techniques to remain resident in memory once their code has been executed and their host program has terminated. This implies that, once a single infected program has been run, the virus potentially can spread to any or all programs in the system. This spreading occurs during the entire work session (until the system is rebooted to clear the virus from memory), rather than during a small period of time when the infected program is executing viral code. These viruses are *resident* or *indirect* viruses, known as such because they stay resident in memory, and indirectly find files to infect as they are referenced

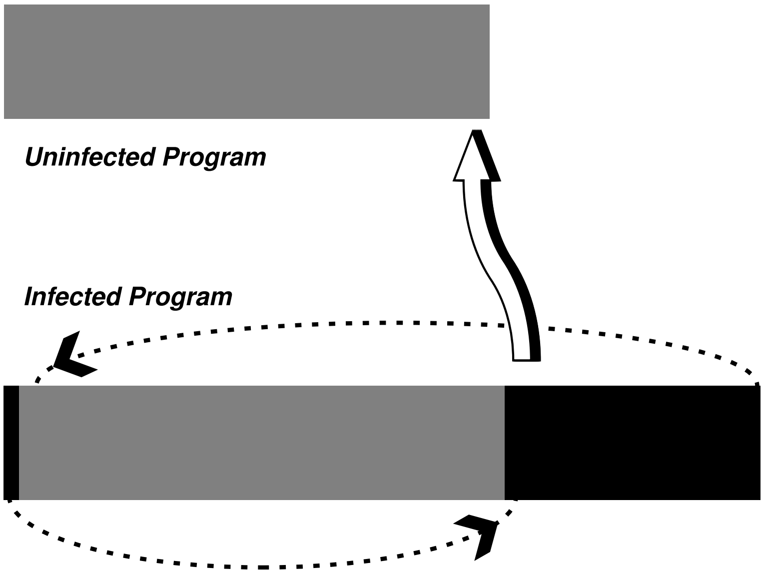

Figure 2: Add-on Virus Infection

by the user. These viruses are also known as TSR (**T**erminate and **S**tay **R**esident) viruses.

If a virus is present in memory after an application exits, how does it remain active? That is, how does the virus continue to infect other programs? The answer for personal computers running software such as MS-DOS is that the virus alters the standard interrupts used by DOS and the BIOS (Basic Input/Output System). The change to the environment is such that the virus code is invoked by other applications when they make service requests.

The PC uses many interrupts (both hardware and software) to deal with asynchronous events and to invoke system functions. All services provided by the BIOS and DOS are invoked by the user storing parameters in machine registers, then causing a software interrupt.

When an interrupt is raised, the operating system calls the routine whose address it finds in a special table known as the *vector* or *interrupt* table. Normally, this table contains pointers to handler routines in the ROM or in memory-resident portions of the DOS (see figure 4). A virus can modify this table so that the interrupt causes viral code (resident in memory) to be executed.

By trapping the keyboard interrupt, a virus can arrange to intercept the CTRL-ALT-DEL soft reboot command, modify user keystrokes, or be invoked on each keystroke. By trapping the `BIOS` disk interrupt, a virus can intercept all `BIOS` disk activity, including reads of boot sectors, or disguise disk accesses to infect as part of a user's disk request. By trapping the `DOS` service

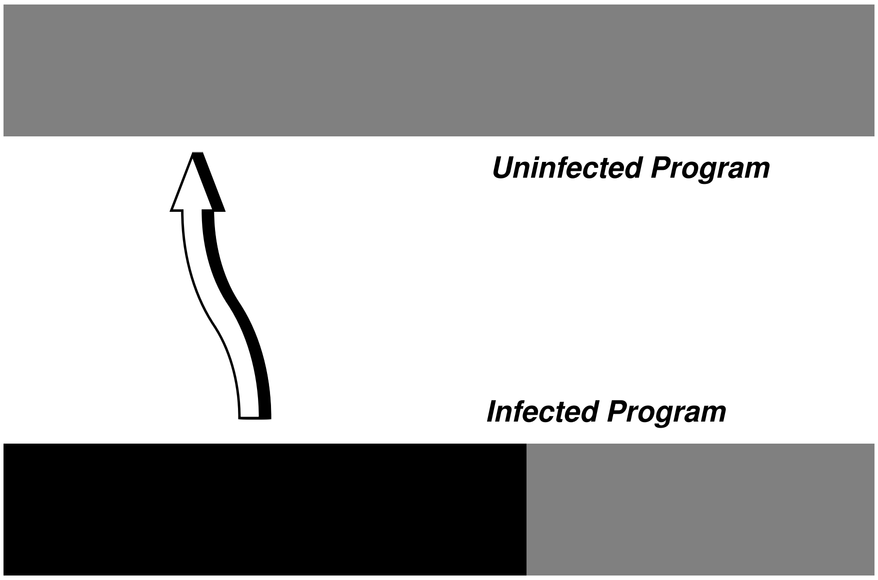

Figure 3: Intrusive Virus Infection

interrupt, a virus can intercept all DOS service requests including program execution, DOS disk access, and memory allocation requests.

A typical virus might trap the DOS service interrupt, causing its code to be executed before calling the real DOS handler to process the request. (See figure 5.)

Once a virus has infected a program or boot record, it seeks to spread itself to other programs, and eventually to other systems. Simple viruses do no more than this, but most viruses are not simple viruses. Common viruses wait for a specific triggering condition, and then perform some activity. The activity can be as simple as printing a message to the user, or as complex as seeking particular data items in a specific file and changing their values. Often, viruses are destructive, removing files or reformatting entire disks. Many viruses are also faulty and may cause unintended damage.

The conditions that trigger viruses can be arbitrarily complex. If it is possible to write a program to determine a set of conditions, then those same conditions can be used to trigger a virus. This includes waiting for a specific date or time, determining the presence or absence of a specific set of files (or their contents), examining user keystrokes for a sequence of input, examining display memory for a specific pattern, or checking file attributes for modification and permission information. Viruses also may be triggered based on some random event. One common trigger component is a counter used to determine how many additional programs the virus has succeeded

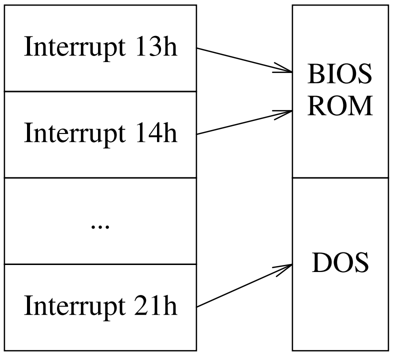

Figure 4: Normal interrupt usage

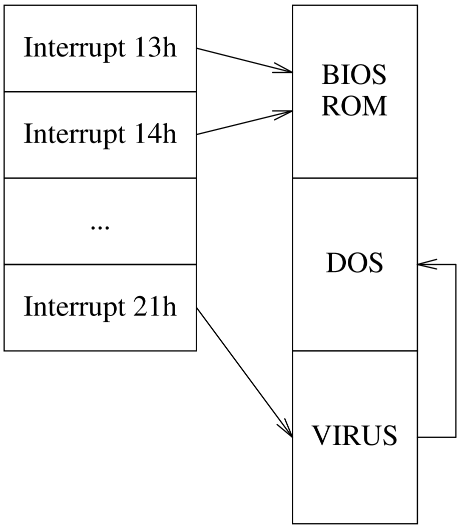

Figure 5: Interrupt vectors with TSR virus

in infecting — the virus does not trigger until it has propagated itself a certain minimum number of times. Of course, the trigger can be any combination of conditions, too.

Computer viruses can infect any form of writable storage, including hard disk, floppy disk, tape, optical media, or memory. Infections can spread when a computer is booted from an infected disk, or when an infected program is run. This can occur either as the direct result of a user invoking an infected program, or indirectly through the system executing the code as part of the system boot sequence or a background administration task. It is important to realize that often the chain of infection can be complex and convoluted. With the presence of networks, viruses can also spread from machine to machine as executable code containing viruses is shared between machines.

Once activated, a virus may replicate into only one program at a time, it may infect some randomly-chosen set of programs, or it may infect every program on the system. Sometimes a virus will replicate based on some random event or on the current value of the clock. The different methods will not be presented in detail because the result is the same: there are additional copies of the virus on the system.

4 Evolution of Viruses

Since the first viruses were written, we have seen what may be classified as five "generations" of viruses. Each new class of viruses has incorporated new features that make the viruses more difficult to detect and remove. Here, as with other classification and naming issues related to viruses, different researchers use different terms and definitions (cf. [9, Appendix 10]). The following list presents one classification derived from a number of these sources. Note that these "generations" do not necessarily imply chronology. For instance, several early viruses (e.g., the "Brain" and "Pentagon" viruses) had stealth and armored characteristics. Rather, this list describes increasing levels of sophistication and complexity represented by computer viruses in the MS-DOS environment.

4.1 First generation: Simple

The first generation of viruses were the simple viruses. These viruses did nothing very significant other than replicate. Many new viruses being discovered today still fall into this category. Damage from these simple viruses is usually caused by bugs or incompatibilities in software that were not anticipated by the virus author.

First generation viruses do nothing to hide their presence on a system, so they can usually be found by means as simple as noting an increase in size of files or the presence of a distinctive pattern in an infected file.

4.2 Second generation: Self-recognition

One problem encountered by viruses is that of repeated infection of the host, leading to depleted memory and early detection. In the case of boot sector viruses, this could (depending on strategy) cause a long chain of linked sectors. In the case of a program-infecting virus, repeated infection may result in continual extension of the host program each time it is reinfected. There are indeed some older viruses that exhibit this behavior.

To prevent this unnecessary growth of infected files, second-generation viruses usually implant a unique *signature* that signals that the file or system is infected. The virus will check for this signature before attempting infection, and will place it when infection has taken place; if the signature is present, the virus will not reinfect the host.

A virus signature can be a characteristic sequence of bytes at a known offset on disk or in memory, a specific feature of the directory entry (e.g., alteration time or file length), or a special system call available only when the virus is active in memory.

The signature presents a mixed blessing for the virus. The virus no longer performs redundant infections that might present a clue to its presence, but the signature does provide a method of detection. Virus sweep programs can scan files on disk for the signatures of known viruses, or even "inoculate" the system by providing the viral signature in clean systems to prevent the virus from attempting infection.

4.3 Third Generation: Stealth

Most viruses may be identified on a contaminated system by means of scanning the secondary storage and searching for a pattern of data unique to each virus. To counteract such scans, some resident viruses employ stealth techniques. These viruses subvert selected system service call interrupts when they are active. Requests to perform these operations are intercepted by the virus code. If the operation would expose the presence of the virus, the operation is redirected to return false information.

For example, a common virus technique is to intercept I/O requests that would read sectors from disk. The virus code monitors these requests. If a read operation is detected that would return a block containing a copy of the virus, the active code returns instead a copy of the data that would be present in an uninfected system. In this way, virus scanners are unable to locate the virus on disk when the virus is active in memory. Similar techniques may be employed to avoid detection by other operations.

4.4 Fourth Generation: Armored

As anti-virus researchers have developed tools to analyze new viruses and craft defenses, virus authors have turned to methods to obfuscate the code of their viruses. This "armoring" includes adding confusing and unnecessary code to make it more difficult to analyze the virus code. The defenses may also take the form of directed attacks against anti-virus software, if present on the affected system. These viruses appeared starting in 1990.

Viruses with these forms of defenses tend to be significantly larger than simpler viruses and thus more easily noticed. Furthermore, the complexity required to significantly delay the efforts of trained anti-virus experts appears to be far beyond anything that has yet appeared.

4.5 Fifth Generation: Polymorphic

The most recent class of viruses to appear on the scene are the polymorphic or self-mutating viruses. These are viruses that infect their targets with a modified or encrypted version of themselves. By varying the code sequences written to the file (but still functionally equivalent to the original), or by generating a different, random encryption key, the virus in the altered file will not be identifiable through the use of simple byte matching. To detect the presence of these viruses requires that a more complex algorithm be employed that, in effect, reverses the masking to determine if the virus is present.

Several of these viruses have become quite wide-spread. Some virus authors have released virus "toolkits" that can be incorporated into a complete virus to give it polymorphic capabilities. These toolkits have been circulated on various bulletin boards around the world, and incorporated in several viruses.

5 Defenses and Outlook

There are several methods of defense against viruses. Unfortunately, no defense is perfect. It has been shown that any sharing of writable memory or communications with any other entity introduces the possibility of virus transmission. Furthermore, Cohen, Adleman, and others have shown proofs that the problem of writing a program to exactly detect all viruses is formally undecidable: it is not possible to write a program that will detect every virus without any error.

Of some help is the observation that it is trivial to write a program that identifies all infected programs with 100% accuracy. Unfortunately, this program must identify *every* (or nearly so) program as infected, whether it is or not! This is not particularly helpful to the user, and the challenge is to write a detection mechanism that finds most viruses without generating an excessive number of false positive reports.

Defense against viruses generally takes one of three forms:

Activity monitors Activity monitors are programs that are resident on the system. They monitor activity, and either raise a warning or take special action in the event of suspicious activity. Thus, attempts to alter the interrupt tables in memory, or to rewrite the boot sector would be intercepted by such monitors. This form of defense can be circumvented (if implemented in software) by viruses which activate earlier in the boot sequence than the monitor code.

They are further vulnerable to virus alteration if used on machines without hardware memory protection — as is the case with all common personal computers.

Another form of monitor is one that emulates or otherwise traces execution of a suspect application. The monitor evaluates the actions taken by the code, and determines if any of the activity is similar to what a virus would undertake. Appropriate warnings are issued if suspicious activity is identified.

Scanners Scanners have been the most popular and widespread form of virus defense. A scanner operates by reading data from disk and applying pattern matching operations against a list of known virus patterns. If a match is found for a pattern, a virus instance is announced.

Scanners are fast and easy to use, but they suffer from many disadvantages. Foremost among the disadvantages is that the list of patterns must be kept up-to-date. In the MS-DOS world, new viruses are appearing by as many as several dozen each week. Keeping a pattern file up-to-date in this rapidly changing environment is difficult.

A second disadvantage to scanners is one of false positive reports. As more patterns are added to the list, it becomes more likely that one of them will match some otherwise legitimate code. A further disadvantage is that polymorphic viruses cannot be detected with scanners.

To the advantage of scanners, however, is their speed. Scanning can be made to work quite quickly. Scanning can also be done portably and across platforms, [17], and pattern files are easy to distribute and update. Furthermore, of the new viruses discovered each week, few will ever become widespread. Thus, somewhat out-of-date pattern files are still adequate for most environments. Scanners equipped with algorithmic or heuristic checking may also find most polymorphic viruses. It is for these reasons that scanners are the most widely-used form of anti-virus software.

Integrity checkers/monitors Integrity checkers are programs that generate checkcodes (e.g., checksums, cyclic redundancy codes (CRCs), secure hashes, message digests, or cryptographic checksums) for monitored files.[20] Periodically, these checkcodes are recomputed and compared against the saved versions. If the comparison fails, a change is known to have occurred to the file, and it is flagged for further investigation. Integrity monitors run continuously and check the integrity of files on a regular basis. Integrity shells recheck the checkcode prior to every execution.[3]

Integrity checking is an almost certain way to discover alterations to files, including data files. As viruses must alter files to implant themselves, integrity checking will find those changes, Furthermore, it does not matter if the virus is known or not — the integrity check will discover the change no matter what causes it. Integrity checking also may find other changes caused by buggy software, problems in hardware, and operator error.

Integrity checking also has drawbacks. On some systems, executable files change whenever the user runs the file, or when a new set of preferences is recorded. Repeated false positive reports may lead the user to ignore future reports, or disable the utility. It is also the case that a change may not be noticed until after an altered file has been run and a virus spread. More importantly, the initial calculation of the checkcode must be performed on a known-unaltered version of each file. Otherwise, the monitor will never report the presence of a virus, probably leading the user to believe the system is uninfected.

Several vendors have begun to build self-checking into their products. This is a form of integrity check that is performed by the program at various times as it runs. If the self-check reveals some unexpected change in memory or on disk, the program will terminate or warn the user. This helps to signal the presence of a new virus quickly so that further action may be taken.

If no more computer viruses were written from now on, there would still be a computer virus problem for many years to come. Of the thousands of reported computer viruses, several hundred are well-established on various types of computers around the world. The population of machines and archived media is such that these viruses would continue to propagate from a rather large population of contaminated machines.

Unfortunately, there appears to be no lessening of computer virus activity, at least within the MS-DOS community. Several new viruses are appearing every day. Some of these are undoubtedly being written out of curiosity and without thought for the potential damage. Others are being written with great purpose, and with particular goals in mind — both political and criminal. Although it would seem of little interest to add to the swelling number of viruses in existence, many individuals seem to be doing exactly that.

6 Viruses as Artificial Life

Now that we know what computer viruses are, and how they spread, we can examine the question of whether they represent a form of artificial life. The first, and obvious, question is "What is life?" Without an answer to this question, we will be unable to say if a computer virus is "alive."

One very reasonable list of properties associated with life was presented in [8]. That list included:

- *Life is a pattern in space-time* rather than a specific material object.
- *Self-reproduction*, in itself or in a related organism.

- *Information storage of a self-representation.*
- *A metabolism* that converts matter/energy.
- *Functional interactions with the environment.*
- *Interdependence of parts*.
- *Stability under perturbations* of the environment.
- *The ability to evolve*.
- *Growth or expansion*

Let us examine each of these characteristics in relation to computer viruses.

6.1 Viruses as patterns in space-time

There is a near match to this characteristic. Viruses are represented by patterns of computer instructions that exist over time on many computer systems. Viruses are not associated with the physical hardware, but with the instructions executed (sometimes) by that hardware. Computer viruses, like all functional computer code, are simply manifestations of algorithms. The algorithms themselves also represent an underlying pattern.

It is questionable if these patterns exist in space, however, unless one extends the definition of space to "cyberspace," as represented by a computer system. The patterns of the viruses are a temporary set of electrical and magnetic field changes in the memory or storage of computer systems. The existence of the virus is only within these patterns of energy. Arguably, the code for each virus could be printed in ink on paper, resulting in a more substantiative existence. That, however, is merely a representation of the true virus, and should not be viewed as existence any more than a picture of a person is itself the person.

6.2 Self-reproduction of viruses

One of the primary characteristics of computer viruses is their ability to reproduce themselves (or an altered version of themselves). Thus, this characteristic seems to be met. One of the key characteristics is their ability to reproduce.

However, it is perhaps more interesting to examine this aspect in light of the agent of reproduction. The virus code is not itself the agent — the computer is. It is questionable if this can be considered sufficient for purposes of classification as artificial life. To do so would imply that (for instance) the blueprints for a Xerox machine are capable of self-reproduction: when outside agents follow the instructions therein, it is possible to produce a new machine that can then be used to make a copy of them. It is not the blueprint (algorithm; virus) that is the agent of change, but the entity that interprets it.

6.3 Information storage of a self-representation

This is the most obvious match for computer viruses. The code that defines the virus is a template that is used by the virus to replicate itself. This is similar to the DNA molecules of what we recognize as organic life.

6.4 Virus metabolism

This property involves the organism taking in energy or matter from the environment and using it for its own activity. Computer viruses use the energy of computation expended by the system to execute. They do not convert matter, but make use of the electrical energy present in the computer to traverse their patterns of instructions and infect other programs. In this sense, they have a metabolism.

Again, however, we are forced to change this view if we examine the case more closely. The expenditure of energy is not by the virus, but by the underlying computer system. If the virus were not active, and an interactive game were being run instead, the same amount of energy would be used. In most systems, even if no program is being run, the energy use remains constant. Thus, we must conclude that viruses do not actually have a metabolism.

6.5 Functional interactions with the virus's environment

Viruses perform examinations of their host environments as part of their activities. They alter interrupts, examine memory and disk architectures, and alter addresses to hide themselves and spread to other hosts. They very obviously alter their environment to support their existence. Many viruses accidentally alter their environment because of bugs or unforeseen interactions. The major portion of damage from all computer viruses is a result of these interactions.

6.6 Interdependence of virus parts

Living organisms cannot be arbitrarily divided without destroying them. The same is true of computer viruses. Should a computer virus have a portion of its "anatomy" excised, the virus would probably cease to function normally, if at all. Few viruses are written with superfluous code, and even so, the working code cannot be divided without disabling the virus.

However, it is interesting to note that the virus can be reassembled later and regain its functional status. If a living organism (as we know them) were to be divided into its component parts for a period of time, then reassembled, it would not become "alive" again. In this sense, computer viruses are more like simple machines or chemical reactions rather than instances of living things.

6.7 Virus stability under perturbations

Computer viruses run on a variety of machines under different operating systems. Many of them are able to compromise (and defeat) anti-virus and copy protection mechanisms. They may adjust on-the-fly to conditions of insufficient storage, disk errors, and other exceptional events. Some are capable of running on most variants of popular personal computers under almost any software configuration — a stability and robustness seen in few commercial applications.

6.8 Virus evolution

Here, too, viruses display a difference from systems we traditionally view as "alive." No computer viruses evolve as we commonly use the term, although it is conceivable that a very complex virus could be programmed to evolve and change. However, such a virus would be so large and complex as to be many orders of magnitude larger than most host programs, and probably bigger than the host operating systems. Thus, there is some doubt that such a virus could run on enough hosts to allow it to evolve. (Note that "evolve" implies a change in function or attributes; polymorphic viruses represent cases of random changes in structure but not functionality.)

Higher-level mutations of viruses do exist, however. There are variants of many known viruses, with over a dozen known for some IBM PC viruses. The variations involved can be very small, on the order of two or three instructions difference, to major changes involving differences in messages, activation, and replication. The source of these variations appears to be programmers (the original virus authors or otherwise) who alter the viruses to avoid anti-viral mechanisms, or to cause different kinds of damage. Polymorphic viruses alter their copies to avoid detection, but the pattern of alteration is ultimately a human product. These changes do not constitute evolution, however.

Interestingly, there is also one case where two different strains of a Macintosh virus are known to interact to form infections unlike the "parents," although these interactions usually produce "sterile" offspring that are unable to reproduce further. This likewise does not appear to be evolution as we know it. [19]

6.9 Growth

Viruses certainly do exhibit a form of growth, in the sense that there are more of them in a given environment over time. Some transient viruses will infect every file on a system after only a few activations. The spread of viruses through commercial software and public bulletin boards is another indication of their wide-spread replication. Although accurate numbers are difficult to derive, reports over the last few years indicate an approximate yearly doubling in the number of systems infected by computer viruses. Clearly, computer viruses are exhibiting significant growth.

6.10 Other behavior

As already noted, computers viruses exhibit "species" with well-defined ecological niches based on host machine type, and variations within these species. These species are adapted to specific environments and will not survive if moved to a different environment.

Some viruses also exhibit predatory behavior. For instance, the DenZuk virus will seek out and overwrite instances of the Brain virus if both are present on the same system. Other viruses exhibit territorial behavior — marking their infected domain so that others of the same type will not enter and compete with the original infection. Some viruses also exhibit self-protective behavior, including camouflage techniques.

It is important to note, however, that none of these characteristics came from the viruses themselves. Rather, each change and addition to virus behavior has been wrought by an outside agency: the programmer. These changes have been in reaction to a perceived need to "enhance" the virus — usually to make it more difficult to find.

It might well be argued that more traditional living organisms may also undergo change from without. As an example, background radiation may cause occasional random mutations. However, programmers are the only source of change to computer viruses, and this distinction is worth noting; other living systems undergo changes to themselves and their progeny without obvious outside agencies.

7 Concluding Comments

Our study of computer viruses at first suggests they are close to what we might define as "artificial life." However, upon closer examination, a number of significant deficiencies can be found. These lead us to conclude that computer viruses are not "alive," nor is it possible to refine them so as to make them "alive" without drastically altering our definition of "life."

To suggest that computer viruses are alive also implies 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.

Undoubtedly, we could adjust our definitions and characteristics to encompass computer viruses or to better exclude them. This illustrates one of the fundamental difficulties with the entire field of artificial life: how to define essential characteristics in such a way as to unambiguously define living systems. Computer viruses provide one interesting example against which such definitions may be tested.

From this, we can observe that computer viruses (and their kin) provide an interesting means of modeling life. For at least this reason, research into computer viruses (using the term in a broader sense, ala Cohen) may be of some scientific interest. By modeling behavior using computer viruses, we may be able to gain some insight into systems with more complex interactions. Research into competition among computer viruses and other software, including anti-viral techniques, is of practical interest as well as scientific interest. Modified versions of viruses such as Thimbleby's Liveware may also prove to be of ultimate value. Research into issues on virus defense methods, epidemeology, and on mutations and combinations also could provide valuable insight into computing.

The problem with research on computer viruses is their threat. True viruses are inherently unethical and dangerous. They operate without consent or knowledge, experience has shown that they cannot be recalled or controlled, and they may cause extensive losses over many years. Even viruses written to be benign cause significant damage because of unexpected interactions and bugs. To experiment with computer viruses is akin to experimenting with smallpox or anthrax microbes — there may be scientific knowledge to be gained, but the potential for disasterous consequences looms large.

In one sense, we use "computer viruses" every day. Editors, compilers, backup utilities, and other common software meet some definitions of viruses. However, their general nature is known to their users, and they do not operate without at least the implied permission of those users. Furthermore, their replication is generally under the close control or observation of their users. It is these differences from the colloquial computer virus that makes the latter so interesting, however. These differences are also precisely what suggest that computer viruses approach a form of artificial life.

If we are to continue to research computer viruses, we need to find fail-safe ways of doing so. This is a major research topic in itself. The danger of creating and accidentally releasing more sophisticated viruses is too great to risk, especially with our increasing reliance on computers in critical tasks. One approach might be to construct custom computing environments for study, different enough from all existing computer systems that a computer virus under study would be completely non-functional outside it. This is an approach similar to what has been taken with Core Wars.[18] Another approach is to only study existing viruses in known environments.

Ultimately, it would be disappointing if research efforts resulted in widespread acceptance of computer viruses as a form of artificial life. It would be especially dangerous to attract the untrained, the careless, and the uncaring to produce them. Already, contests have been announced for virus writers to produce a "useful" or "shortest" virus. Self-reproducing code is easier to write than to control, and encouraging its production in uncontrolled environments is irresponsible; accidents happen all too frequently with computers.

The origin of most computer viruses is one of unethical practice. Viruses created for malicious purposes are obviously bad; viruses constructed as experiments and released into the public domain would likewise be unethical, and poor science besides: experiments without controls, strong hypotheses, and the consent of the subjects. Facetiously, I suggest that if computer viruses evolve into something with artificial consciousness, this might provide a doctrine of "original sin" for their theology.

More seriously, I would suggest that there is something of great importance already to be learned from the study of computer viruses: the critical realization that experimentation with systems in some ways (almost) alive can be hazardous. Computer viruses have caused millions of dollars of damage and untold aggravation. Some of them have been written as harmless experiments that "got away," and others as malicious mischief. A great many of them have firmly rooted themselves in the pool of available computers and storage media, and they are likely to be frustrating users and harming systems for years to come. Similar but considerably more tragic results could occur from careless experimentation with organic forms of artificial life. 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. This is a lesson we all would do well to learn.

References

[1] Leonard Adleman. An abstract theory of computer viruses. In *Lecture Notes in Computer Science, vol 403*. Springer-Verlag, 1990.

[2] Fred Cohen. *Computer Viruses*. PhD thesis, University of Southern California, 1985.

[3] Frederick B. Cohen. *A Short Course on Computer Viruses*. ASP Press, Pittsburgh, PA, 1990.

[4] Frederick B. Cohen. Friendly contagion: Harnessing the subtle power of computer viruses. *The Sciences*, pages 22–28, Sep/Oct 1991.

[5] Peter J. Denning, editor. *Computers Under Attack: Intruders, Worms and Viruses*. ACM Press (Addison-Wesley), 1990.

[6] Tom Duff. Experiences with viruses on Unix systems. *Computing Systems*, 2(2), Spring 1989.

[7] Mark W. Eichin and Jon A. Rochlis. With microscope and tweezers: an analysis of the internet virus of november 1988. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 326–343, Oakland, CA, May 1989. IEEE-CS.

[8] J. Doyne Farmer and Alletta d'A. Belin. Artificial life: The coming evolution. In *Proceedings in Celebration of Murray Gell-Man's 60th Birthday*. Cambridge University Press, 1990. To appear.

[9] David Ferbrache. *A Pathology of Computer Viruses*. Springer-Verlag, 1992.

[10] Christopher V. Feudo. *The Computer Virus Desk Reference*. Business One Irwin, Homewood, IL, 1992.

[11] Philip Fites, Peter Johnson, and Martin Kratz. *The computer virus crisis*. Van Nostrand Reinhold, 2nd edition, 1992.

[12] David Gerrold. *When Harlie Was One*. Doubleday, Garden City, NY, 1972.

[13] William Gibson. *Neuromancer*. Ace/The Berkeley Publishing Group, 1984.

[14] Harold Joseph Highland, editor. *Computer Virus Handbook*. Elsevier Advanced Technology, 1990.

[15] Lance J. Hoffman, editor. *Rogue Programs: Viruses, Worms, and Trojan Horses*. Van Nostrand Reinhold, New York, NY, 1990.

[16] Jan Hruska. *Computer Viruses and Anti-Virus Warfare*. Ellis Horwood, Chichester, England, 1990.

[17] Sandeep Kumar and Eugene H. Spafford. A generic virus scanner in C++. In *Proceedings of the 8th Computer Security Applications Conference*, pages 210–219, Los Alamitos CA, December 1992. ACM and IEEE, IEEE Press.

[18] Steven Levy. *Artificial Life: The Quest for a New Creation*. Pantheon, New York, NY, 1992.

[19] John Norstad. *Disinfectant On-line Documentation*. Northwestern University, 1.8 edition, June 1990.

[20] Yisrael Radai. Checksumming techniques for anti-viral purposes. *1st Virus Bulletin Conference*, pages 39–68, September 1991.

[21] Deborah Russell and Sr. G. T. Gangemi. *Computer Security Basics*. O'Reilly & Associates, Cambridge, MA, 1991.

[22] Donn Seeley. Password cracking: A game of wits. *Communications of the ACM*, 32(6):700–703, June 1990.

[23] John F. Shoch and Jon A. Hupp. The 'worm' programs—early experiments with a distributed computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[24] Alan Solomon. *PC VIRUSES Detection, Analysis and Cure*. Springer-Verlag, London, 1991.

[25] Eugene H. Spafford. An analysis of the internet worm. In C. Ghezzi and J. A. McDermid, editors, *Proceedings of the 2nd European Software Engineering Conference*, pages 446–468. Springer-Verlag, September 1989.

[26] Eugene H. Spafford. The internet worm: Crisis and aftermath. *Communications of the ACM*, 32(6):678–687, June 1989.

[27] Eugene H. Spafford. The internet worm program: an analysis. *Computer Communication Review*, 19(1):17–57, January 1989. Also issued as Purdue CS technical report TR-CSD-823.

[28] Eugene H. Spafford. Computer viruses: A form of artificial life? In D. Farmer, C. Langton, S. Rasmussen, and C.Taylor, editors, *Artificial Life II, Studies in the Sciences of Complexity*, pages 727–747. Addison-Wesley, Redwood City, CA, 1991. Proceedings of the second conference on artificial life.

[29] Eugene H. Spafford. Response to Fred Cohen's "contest". *The Sciences*, page 4, Jan/Feb 1992.

[30] Eugene H. Spafford. Computer viruses. In John Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[31] Eugene H. Spafford, Kathleen A. Heaphy, and David J. Ferbrache. *Computer Viruses: Dealing with Electronic Vandalism and Programmed Threats*. ADAPSO, Arlington, VA, 1989.

[32] Brad Stubbs and Lance J. Hoffman. Mapping the virus battlefield. In Hoffman [15], chapter 12, pages 143–157.

[33] I. H. Witten, H. W. Thimbleby, G. F. Coulouris, and S. Greenberg. Liveware: A new approach to sharing data in social networks. *International Journal of Man-Machine Studies*, 1990.