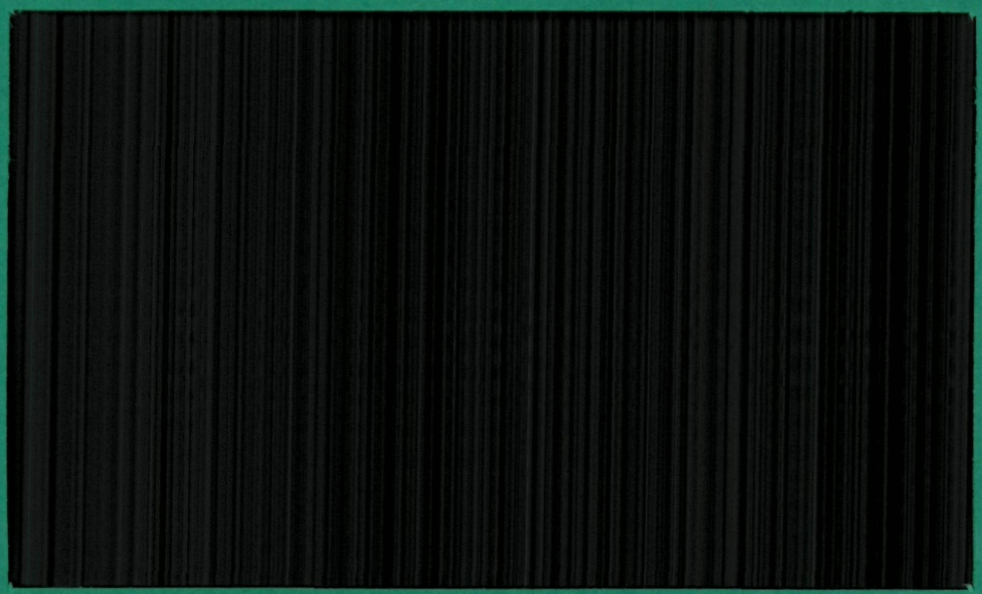
DARTMOUTH COLLEGE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

HANOVER, NEW HAMPSHIRE 03755-3551



TECHNICAL REPORT

AN OVERVIEW OF COMPUTER VIRUSES IN A RESEARCH ENVIRONMENT

Matt Bishop

Technical Report PCS-TR91-156

An Overview of Computer Viruses in a Research Environment

Matt Bishop

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

ABSTRACT

The threat of attack by computer viruses is in reality a very small part of a much more general threat, specifically attacks aimed at subverting computer security. This paper examines computer viruses as malicious logic in a research and development environment, relates them to various models of security and integrity, and examines current research techniques aimed at controlling the threats viruses in particular, and malicious logic in general, pose to computer systems. Finally, a brief examination of the vulnerabilities of research and development systems that malicious logic and computer viruses may exploit is undertaken.

1. Introduction

A *computer virus* is a sequence of instructions that copies itself into other programs in such a way that executing the program also executes that sequence of instructions. Rarely has something seemingly so esoteric captured the imagination of so many people; magazines from *Business Week* to the *New England Journal of Medicine* [37][46][57][69][133], books [19][21][30][38][48][64]-[79][86][105][122], and newspaper articles [81][88][89][91][111] have discussed viruses, applying the name to various types of malicious programs.

As a result, the term "computer virus" is often misunderstood. Worse, many who do understand it have misperceptions about the vulnerabilities of computer systems, for example believing that conventional security mechanisms can prevent virus infections, or are flawed because they cannot. One purpose of this paper is to debunk many of these myths and relate computer viruses to current research and methods in computer security.

The second purpose arises from the analysis of applying conventional computer security mechanisms to viruses. Attacks involving computer viruses combine well-known techniques, so to believe that computer viruses are unique and unrelated to any other type of attack is a fallacy. Existing computer security mechanisms were not designed specifically to counter computer viruses,

This work was supported by grants NAG2-328 and NAG2-628 from the National Aeronautics and Space Administration to Dartmouth College.

but encompass many of the features used by computer viruses. While those mechanisms cannot prevent computer virus infections any more than they can prevent all attacks, they can impede a virus' spread as well as make the introduction of a computer virus difficult, just as they can limit the damage done in an attack, or make a successful attack very difficult. This paper tries to show the precise impact of many conventional security mechanisms on computer viruses by placing the virus within a more general framework.

Because the probability of encountering a computer virus and the controls available to deal with it vary widely among different environments, this paper confines itself to that environment consisting of computers running operating systems designed for research and development, such as the UNIX[1] operating system, the VAX/VMS[2] operating system, and so forth. There is already a wealth of literature on computer viruses within the personal computing world (for example, see [32][59][62][119]), and a simple risk analysis (upon which we shall later elaborate) indicates that systems designed for accounting, inventory control, and other primarily business oriented operations are less likely to be attacked using computer viruses than by other methods. So, while some of the following discussion may be fruitfully applied to computer systems in those environments (for example, see [1]), many of the underlying assumptions put the usefulness of other parts of the discussion into question when dealing with other environments.

First, we shall review what a computer virus is, and analyze the properties that make it a threat to computer security. Next, we present a brief history of computer viruses and consider whether their threat is relevant to research and development systems, and if so, how. After exploring some of the research in secure systems that show promise for coping with viruses, we examine several specific areas of vulnerability in research-oriented systems. We conclude with a brief summary.

Given the variety of definitions of "computer virus," we now present the standard one.

2. What is a Computer Virus?

Computer viruses do not appear spontaneously [24]; an *attacker* must first write, and then introduce, a virus to a computer system. Once there, the virus will then attempt to take some action, such as acquire privileges or damage or destroy information. The simplest way to do this is to have an authorized person perform the action. A technique that first appeared not in the modern age but

1. UNIX is a Registered Trademark of AT&T Bell Laboratories.
2. VAX and VMS are trademarks of Digital Equipment Corporation.

in antiquity (see sidebar 1) does just that: a *Trojan horse* is something that performs a stated function while secretly performing another, unstated and usually undesirable one. For example, suppose a file used to boot a microcomputer contained a Trojan horse inserted in unused storage space on the floppy disk. When the microcomputer boots, it would execute the Trojan horse, which could erase the disk. Here, the overt function is to provide a basic operating system; the covert function is to erase the disk.

D. Edwards first applied the term "Trojan horse" to these attacks [4], which proved very effective (see for example [96][98]). Many security studies in the late 1960's and early 1970's analyzed them, and one such study [71] described a Trojan horse that reproduces itself (a *replicating Trojan horse*). If such a program *infects* another by inserting a copy of itself into another file or process, it is a *computer virus*. (See sidebar 2; Leonard Adelman first called programs with the infection property "viruses" in a computer security seminar in 1983 [24].)

A computer virus infects other entities during its *infection phase* and then performs some additional (possibly null) actions during its *execution phase* This nomenclature has created a controversy about whether a virus is a Trojan horse [42][66] or something else that can carry a Trojan horse within it [24][94]. Many view the infection phase as part of the "covert" action of a Trojan horse, and consequently consider the virus to be a form of the Trojan horse. Others treat the infection phase as "overt" and so distinguish between the virus and the Trojan horse, since a virus may infect and perform no covert action. Of course, everyone agrees that a virus may perform covert actions during its execution phase.

A computer virus is an instance of the more general class of *malicious logic* or *malicious programs*; other examples are the *worm*, which copies itself from computer to computer[1]; the *bacterium*, which copies itself to such an extent that it absorbs all available resources of the host computer; the *logic bomb*, which is executed when certain events occur (such as a specific date, like Friday the 13th), and the Trojan horse [37]. But none of these satisfies the infection property; any may transport a virus, but since they do not insert themselves into other programs or processes, they are not themselves viruses.

A key point is that malicious logic in general, and computer viruses in particular, take advantage of the user's rights to perform their functions; the virus will spread only as the user's rights will allow it, and can only take those actions that the user may take. They can perform any action

1. Originally, a worm was simply a distributed computation [112]; it is now most often used in the above sense.

the user can legitimately take because operating systems cannot distinguish between actions the user intended and actions the user did not intend. And as the programs containing the viruses are shared among users, the viruses spread among those users [24][94]. In particular, given a system with an infected initial state, a virus will infect files until all programs writable by any infected program are themselves infected [53].

3. Malicious Logic, Computer Viruses, and Computer Security

The basis for dealing with any attack on a computer is the site's *security policy*, which describes how users may access the computer system or information on it. The policy's goal depends largely on how the system is to be used. *Military system security policies* deal primarily with disclosure of information, whereas *commercial security policies* deal primarily with the integrity of data on a system.

Mechanisms designed to enforce both types of security policies partition the system into *protection domains* which define the set of objects that processes may access. *Mandatory access controls* prevent processes from crossing protection domain boundaries, and many systems designed to meet rigid security policies provide them. *Discretionary access controls* permit processes to cross domain boundaries using a limited set of commands if both the process identity and information associated with the object to be accessed allow that access.

Policies using mandatory access controls to prevent disclosure define a linear ordering of security levels, and a set of classes into which information is placed. Each entity's security classification is defined by the pair (*security level, set of classes*); the security classification of entity *A* *dominates* that of entity *B* if *A*'s security level is at least that of *B* and *A*'s set of classes contains all elements of *B*'s set of classes. Then the controls usually enforce some variant of the *Bell-LaPadula model* [9]: a subject may read an object only if the subject's security classification dominates that of the object (the *simple security property*) and a subject may modify an object only if the object's security classification dominates that of the subject (the **-property* or the *confinement property*). Hence subjects may obtain information only from entities with "lower" security classifications, and may disclose information only to entities with a "higher" security classification. These controls limit malicious logic designed to disclose information to the relevant protection domain; they do not limit malicious logic designed to corrupt information in "higher" security classifications.

Policies using discretionary access controls to limit disclosure assume that all processes of a given identity act with the authorization of that identity, and do not inhibit processes with mali-

cious side effects. When a program containing malicious logic is executed, the malicious logic executes with the same identity as that user's legitimate processes. As the protection mechanism has no way to distinguish between acts done for the user and acts done for the attacker by the malicious logic, it allows *any* such action, including infection by a computer virus. So the virus can spread.

Policies using mandatory access controls to limit modification of entities often implement the mathematical dual of the multilevel security model described above. Multilevel integrity models define integrity levels and classes analogous to those of the multilevel security models; then controls may enforce the *Biba integrity model* [11], which allows a subject to read an entity only if the entity's integrity classification dominates that of the subject (the *simple integrity property*), and a subject to modify an entity only if the subject's integrity classification dominates that of the entity (the *integrity confinement property*). This prevents a subject from modifying data or other programs at a higher integrity level, and a subject from relying on data or other programs at a lower integrity level; specifically, malicious logic can only damage those entities with lower or equal integrity classifications.

Lipner has proposed using the multilevel disclosure model to enforce multilevel integrity by assigning classifications and levels to appropriate user communities [83]; however, he notes that malicious logic could "write up" and thereby infect programs or alter production data and code. Clark and Wilson have proposed an alternate model [23], in which data and programs are manipulated by well-defined "transformation procedures," these procedures having been certified by the system security officer as complying with the site integrity policy. As only these procedures can manipulate production data or programs, computer viruses could only propagate among production programs if a transformation procedure which contains one is itself certified to conform to the integrity policy; presumably, the system security officer would detect any such transformation procedure and refuse to certify it.

Policies using discretionary access controls to limit modification of entities make the same assumptions as security policies using discretionary access controls, with similar results.

Systems implementing multilevel security and integrity policies usually allow exceptions to the stated policy; these exceptions are permitted only for entities trusted not to abuse the privilege, and are necessary for the smooth operation of the computer system. An interesting observation is that the usefulness of whatever security model the system implements depends to a very great extent on these exceptions; for should a trusted entity attempt to abuse its power to deviate from the strict policy, little can be done. Hence the statements describing the effects of the controls

on malicious logic above should be read with this in mind; they apply only to the model, and must be suitably modified for those situations in which a security policy allows (trusted) entities to violate the policy.

The two phases of a computer virus' execution illustrate this. The infection phase results in a program being altered, which may violate the strict interpretation of the model of a site's integrity policy (but be possible due to an exception to those rules, as discussed above). The execution phase results in the disclosure of some information across protection domain boundaries, again forbidden by the strict interpretation of the model of the site's security policy, but possible because of an allowed exception. So the virus spreads more rapidly because of the exceptions.

The insertion of malicious logic, for example by infection, causes the altered program to deviate from its specification. If this is considered an "error" as well as a breach of security, fault-tolerant computer systems, which are designed to continue reliable operation when errors occur, could constrain malicious logic. Designers of reliable systems place emphasis on both recovery and preventing failures [103]; however, if malicious logic discloses information or gives away rights, or controls other critical systems (such as life support systems), recovery may not be possible. So the areas of reliability and fault-tolerance are relevant to the study of malicious logic, but those areas of fault recovery are less so.

In the most general case, whether a given program will infect another is undecidable [2][24], so programs that look for virus infections must check characteristics of known viruses rather than rely on a general infection detection scheme. Further, viruses can be programmed to mutate, and hence be able to evade those agents, which in turn can be programmed to detect the mutations; and in the general case, whether or not one virus mutated to produce another virus is also undecidable [29].

4. The Threat of Computer Viruses

Some history of computer viruses and replicating Trojan horses in research and development environments is appropriate at this point. One of the earliest documented replicating Trojan horses was a version of the game program *animal*; whenever anyone played it, it created a second copy of itself. The program spread to many different computers as users made tapes of it when they moved. A later version would locate and delete one copy of the first version, and then create *two* copies of itself. Because it spread even more rapidly than the first version, this later program supplanted the first entirely. Then after a prespecified time, whenever anyone ran a copy of the later

version, it deleted itself after the game ended [39].

Ken Thompson created a far more subtle replicating Trojan horse when he rigged a compiler to break login security [104][125]. When the compiler compiled the login program, it would secretly insert instructions to cause the resulting executable program to accept a fixed, secret password as well as a user's real password. Also, when compiling the compiler, the Trojan horse would insert commands to modify the login command into the resulting executable compiler. He then compiled the compiler, deleted the new source, and reinstalled the old source. Since it showed no traces of being doctored, anyone examining the source would conclude the compiler was safe. Fortunately, Thompson took some pains to ensure that it did not spread further, and it was erased when someone copied another version of the executable compiler over the sabotaged one. Thompson's point was that "no amount of source-level verification or scrutiny will protect you from using untrusted code" ([125], p. 763); and this bears remembering, especially given the reliance of many security techniques relying on humans certifying programs to be free of malicious logic.

In 1983, Fred Cohen conducted a series of experiments to determine if viruses could spread readily on non-PC systems. He designed his virus to acquire privileges rather than delete files; on a VAX-11/750 running UNIX, the originator of the virus placed an infected program onto the system bulletin board. He obtained all system rights within half an hour on the average; the longest time needed was an hour, the least, under 5 minutes. Further, because the virus ran so swiftly that it did not degrade response time noticeably, most users never knew the system was under attack. But even those who did know were infected anyway. In 1984 an experiment involving a UNIVAC 1108 showed that viruses could spread throughout that system too. Viruses were also written for other systems (TOPS-20, VAX/VMS, and a VM/370 system) but testing their effectiveness was forbidden. Cohen's experiments indicated that the security mechanisms of those systems did little if anything to inhibit computer virus propagation [24][25].

In 1987, Tom Duff experimented on UNIX systems with a small virus that copied itself into executable files. The virus was not particularly virulent, but when Duff placed 48 infected programs on the most heavily used machine in the computing center, the virus spread to 46 different systems and infected 466 files, including at least one system program on each computer system, within eight days. Duff did not violate the security mechanisms in any way when he seeded the original 48 programs [43]. Duff also wrote a virus in a language used by a command interpreter common to most UNIX systems, and so would run on machines of radically different architectures. His program disproved a common fallacy [48] by showing that computer viruses need not be ma-

chine dependent, and can spread to many systems of varying architectures.

On November 2, 1988, a program combining elements of a computer worm and a computer virus targeting Berkeley and Sun UNIX-based computers entered the Internet; within hours, it had rendered several thousand computers unusable [44][45][106][114][115][120][121][123]. Among other techniques, this program used a virus-like attack to spread: it inserted some instructions into a running process on the target machine and arranged for those instructions to be executed. To recover, these machines had to be disconnected from the network, rebooted, and several critical programs changed and recompiled to prevent re-infection. Worse, the only way to determine if the program had other malicious side effects (such as deleting files) was to disassemble it. Fortunately, its only purpose turned out to be to propagate. Infected sites were extremely lucky that the worm[1] did not infect a system program with a virus designed to delete files, or did not attempt to damage attacked systems; at most, that would have taken one or two extra lines of code. Since then, there have been several incidents involving worms [56][63][123].

In general, though, computer viruses and replicating Trojan horses have been laboratory experiments rather than attacks from malicious or careless users. This leads to the question of risk analysis: do the benefits gained in defending against computer viruses offset the costs of recovery and the likelihood of being attacked?

As worded, the above question implies that the mechanisms defending against computer viruses are useful *only* against computer viruses. However, computer viruses are only a particular example of programs containing malicious logic, and all such programs have various characteristics in common. Hence defenses which examine ways to strengthen access controls to prevent illicit access, or which prevent or detect the alteration of other files, work not only against computer viruses but also against more conventional forms of malicious logic, such as Trojan horses. So, to rephrase the question: do the benefits gained in defending against malicious logic offset the costs of recovery and the likelihood of being attacked?

Because this paper focuses primarily on computer viruses, we shall not delve into the history of the use of malicious logic in general. Suffice it to say that attacks using such techniques are well known and have been used often (see both [96] and [98] for descriptions of such incidents), and use of mechanisms to inhibit them is generally agreed to be worthwhile.

---

1. We use the conventional terminology of calling this program a "computer worm" because its dominant method of propagation was from computer system to computer system. Others, notably [44], have labelled it a "computer virus" using a taxonomy more firmly grounded in biology than the conventional one.

5. Current Research in Malicious Logic and Computer Viruses

Gödel's Undecidability and Incompleteness theorems imply that "for any [interesting] consistent theory ..., its consistency cannot be proven within the theory" ([87], p.118). Similarly, the effectiveness of any security mechanism cannot be assured using only that mechanism. The security it provides depends also on the security of the underlying base on which the mechanism is implemented, and the correctness of the necessary checking done at each step. No matter to what granularity the security is checked, the results will assume that some component of the base or of the checking process is secure or correct – and if this trust is misplaced the mechanism will not be secure. For this reason, "secure" is a relative notion, as is "trust," and mechanisms to enhance computer security attempt to balance the cost of the mechanism with the level of security desired and the degree of trust in the base that the site accepts as reasonable.

Research dealing with malicious logic assumes the interface, software, and/or hardware used to implement the proposed scheme performs exactly as desired. Here the trust is in the underlying computing base, the implementation, and (if done) the verification. If this trust is misplaced and the computing base permits the corruption of the mechanism, then it will be worthless.

Current research uses specific properties of computer viruses to detect and limit their effects. Because of the fundamental nature of these properties, these defenses work equally well against most other forms of malicious logic.

5.1. Computer Viruses Acting as Both Data and Instructions

Techniques exploiting this property treat all programs as type "data" until some certifying authority changes the type to "executable" (instructions). Both new systems designed to meet strong security policies and enhancements to existing systems use this method.

The Logical Coprocessor Kernel or LOCK (formerly the Secure Ada Target or SAT) [16][58][109][110], designed to meet the highest level of security under the Department of Defense criteria [41], allows users to share segments of instructions. To limit propagation of viruses through this sharing, only one copy of the instructions of the shared routine is in memory. A master directory, accessible only to a trusted hardware controller, associates with each procedure a unique owner, and each user specifies who is trusted not to infect that user. Before executing any procedure, the dynamic linker checks that the user executing the procedure trusts the procedure's owner [15].

Note that if a virus infects a trusted user's procedures, the infection could spread to all those who trust the infected user. Boebert and Kain [17] propose labeling subjects and objects with *types*.

Once compiled, programs have type "data," and cannot be executed until a sequence of specific, auditable events changes the type to "executable." After that, the program cannot be modified. This scheme recognizes that viruses treat programs as data (when they infect them by changing the file's contents) and as instructions (when the program executes and spreads the virus), and rigidly separates the two. The Argus Security Model [3] uses the same principle.

Duff [43] has suggested a variant for UNIX-based systems. Noting that users with execute permission for a file usually also have read permission, he proposes that files with execute permission be of type "executable," and those without it be of type "data." Unlike the LOCK, "executable" files could be modified but doing so would change the type to "data." If the certifying authority were the omnipotent user, the virus could spread only if run as that user. Of course, a virus could infect a library (which is not executable), and from there spread to any program using that library, so the certification procedure must apply to libraries and other related files as well as executables.

Both the LOCK scheme and Duff's proposal trust that the administrators will never certify a program containing malicious logic (either by accident or deliberately), and that the tools used in the certification process are not themselves corrupt.

5.2. Viruses Assuming the Identity of a User

Among the many enhancements to discretionary access controls are suggestions to allow the user to reduce the associated protection domain [28][69][118][132]; to base access to files on some characteristic of the command or program [26][77], possibly including subject authorizations as well [24]; and to use a knowledge-based subsystem to determine if a program makes reasonable file accesses [70]. Such mechanisms trust the users to take explicit action to limit their protection domains sufficiently; or trust tables to describe the programs' expected actions sufficiently for the mechanism to apply those descriptions, and the mechanism to handle commands with no corresponding table entries effectively; or they trust specific programs and the kernel, when those would be the first programs a virus would attack.

Mechanisms allowing users to specify semantics for file accesses [10][34] may prove useful for dealing with viruses in some contexts, for example protecting a limited set of files.

5.3. Viruses Crossing Protection Domain Boundaries by Sharing.

As viruses propagate among protection domains when users in those domains share pro-

grams or data, preventing such sharing will inhibit viruses from spreading. One proposal [134] suggests placing programs to be protected at the lowest possible level of an implementation of a multilevel security policy. Since the mandatory access controls will prevent those processes from writing to objects at lower levels (by the *-property), any process can read the programs but no process could write to them. Such a scheme would have to be combined with an integrity model to provide protection against viruses to prevent both disclosure and file corruption; such a combination would work reasonably well. Carrying this idea to its extreme would result in isolation of each domain; as Cohen has pointed out [24], since sharing in such a system is not possible, no viruses can propagate. Of course, the usefulness of such systems would be minimal.

5.4. Viruses Altering Files

Mechanisms using *manipulation detection codes* (or *MDCs*) apply some function to a file to obtain a set of bits called the *signature block* and encrypt that block. The user (or the operating system) can then check for infections by recomputing the signature block and reencrypting it; if the result differs from the stored signature block, the file has changed [82][92]. A similar scheme would embed such MDCs in data or programs distributed over networks; sites would keep encrypted audit trails to allow tracing changes to the site on which they occurred. Note that these efforts also detect non-virus related alterations, and hence are suitable for many other purposes as well.

All assume the executable file does not contain a virus before it is signed. Page [97] has suggested expanding the model in [16] to include the software development process (in effect limiting execution domains for each development tool and user) to ensure software is not contaminated during development. Pozzo and Grey [101][102] have proposed a mechanism to incorporate a measure of this trust into a system using Biba's integrity model. They assume a tamper-proof kernel and software, and have different classes of signed executable programs. *Credibility ratings* (Biba's "integrity levels") assign a measure of trustworthiness on a scale of 0 (unsigned) to *N* (signed and formally verified), based on the origin of the software. Trusted file systems contain only signed executables with the same credibility level. Associated with each user (subject) is a *risk level* that starts out as the highest credibility level but that the user may lower. Execute a program with a credibility level lower than that user's risk level requires a special "run-untrusted" command. This scheme has been implemented on the distributed operating system LOCUS [100].

All integrity-based schemes rely on software which if infected may fail to report tampering. Performance will be affected as encrypting the file or computing the signature block may take a

significant amount of time. The encrypting key must be secret, for if not then a virus can easily alter a signed file undetectably. Even with a public-key cryptosystem, access to the enciphering key compromises the system because the virus can simply overwrite the contents of the executable file with whatever it pleases.

Network implementations of MDC-based mechanisms require a secure out-of-band public key distribution method as well; for if the key distribution mechanism used the same paths as the data transmission, a malicious site (or set of cooperating malicious sites) could alter the data or program being sent, recompute the signature block and sign it with its own (bogus) private key, and then transmit the data; when the public key were requested, it would simply send the one corresponding to the (bogus) private key. The more general (non-network) software distribution problem has similar requirements [33].

Anti-virus agents check files for specific viruses and if present either warn the user or attempt to "cure" the infection by removing the virus. Many such agents exist for personal computers, but since each must look for a particular virus or set of viruses, they are very specific tools and, because of the undecidability results stated earlier, cannot deal with viruses not yet analyzed.

5.5. Viruses Performing Actions Beyond Specification

Fault-tolerant techniques keep systems functioning correctly when the software or hardware fails to perform to specification. Joseph and Avižienis have suggested treating a virus' infection and execution phases as errors. To continue functioning correctly means to prevent the spread (and associated actions) of the virus. The first such proposal [67][68] breaks programs into sequences of non-branching instructions, and checksums each sequence, storing the results in encrypted form. When the program is run, the processor recomputes checksums, and at each branch, a co-processor compares the computed checksum to the encrypted checksum; if they differ, an error (which may be an infection) has occurred. Later proposals advocate checking each instruction [33]. These schemes raise issues of key management and protection, as well as how much the software managing keys, transmitting the control flow graph to the co-processor, and implementing the recovery mechanism, may be trusted.

A proposal based on *N-Version Programming* [5] requires implementing several different versions of a program, running them concurrently and periodically checking intermediate results against each other. If they disagree, the value assumed correct is the intermediate value that a majority of the programs have obtained, and the programs with a different value are malfunctioning

(possibly due to malicious logic). Unfortunately, it assumes that a majority of the programs are not infected, and that the underlying operating system is secure. Also, the issue of the efficacy of N-version programming is highly questionable [73]. Despite claims that the method is feasible [6][22], detecting the spread of a virus would require voting upon each file system access; to achieve this level of comparison, the programs would all have to implement the same algorithm, which defeats the purpose of using N-version programming [74].

5.6. Viruses Altering Statistical Characteristics

Other proposals suggest examining the appearance of program for identical sequences of instructions or byte patterns [66][134]; however, this requires a high number of comparisons and as most programmers use libraries of common routines or re-use code [72], the number of false alarms would be staggering. A similar approach assumes that programmers have their own individual styles of writing programs, that the executable programs generated by the compilers will reflect these styles, and that a *coding style analyzer* can distinguish these styles from one another [134]. A virus might be present if a program appears to have more programmers than were known to have worked on it, or if one particular programmer appears to have worked on many different and unrelated programs. These assumptions must be better established before this method is practicable. A third proposal compares an object file with the source to find conditionals in the former not corresponding to any in the latter; in some cases, these may indicate infection [51]. A fourth suggests designing a filter to detect, analyze, and classify all modifications that a program will make as ordinary or suspicious [31].

Finally, Dorothy Denning has suggested using an intrusion-detection expert system to detect viruses by looking for increases in the size of files, increases in the frequency of writing to executable files, or alterations in the frequency of executing a specific program in ways not matching the profile of users spreading the infection [36]. Several groups have implemented her model [8][84][124] and results indicate that such a system can detect anomalies without noticeably degrading the monitored computer. However, the experiments did not attempt to validate any claims about detecting viruses, so her idea remains simply an interesting proposal.

These research proposals are for the most part merely ideas or suggestions; those that are being implemented are either targeted for specific architectures or are in the very early stages of development. This state of affairs is unsettling for the managers and administrators of existing systems, who need to take some action to protect their users and systems.

6. Vulnerabilities of Existing Research-Oriented Systems

The vulnerabilities exploited by a computer virus can also be exploited by other forms of malicious logic, and unless the purpose of the attack is to cause mischief, the simpler forms of malicious logic are more effective because they are usually simpler. Rather than describe appropriate countermeasures, we simply note that these will differ from environment to environment, and no such list (or even set of lists) can accurately reflect the idiosyncracies of all the different research and development systems; in short, providing such a generic list could provide a very false sense of security.

This section discusses the areas of vulnerability; while we emphasize computer viruses throughout, these same vulnerabilities can be exploited by Trojan horses, computer works, and other forms of malicious logic. We leave it to the reader to formulate appropriate techniques to detect or hinder attacks exploiting each area. (Sidebar 3 offers a starting point for UNIX-based systems.)

6.1. Computing Base

Users assume that the computer system provides a set of trustworthy tools for compiling, linking and loading, and running programs. In a system with a trusted computing base, the "trusted tools" are part of that base; in other systems, the notion of trust is the user's estimate of the quality of the tools available [27] and the working environment. If the estimates are incorrect, the system may be subverted.

Even systems with security enhancements are vulnerable. One version of the UNIX operating system with security enhancements was breached when a user created a version of the directory lister, with a Trojan horse, in his home directory. He then requested assistance from the system operator, who changed to the user's home directory, and listed the names of the files in it. As the command interpreter checked for commands in the current working directory and *then* in the system directories, the user's doctored lister, not the system lister, was executed. The Trojan horse had privileges sufficient to read a protected file [117].

Ken Thompson spoke about this vulnerability when he pointed out that using any computer system involves a degree of trust. In the above, the system administrator trusted the command interpreter to look for system programs before executing programs in users' directories. Other examples include trusting that the login banner being presented is actually from the login program and not from a user's program which will record passwords [55], or that page faults cannot be detected while checking passwords one character at a time [78].

6.2. Sharing Hardware and Software

Intimately bound with the notion of trust is the ability to share. When many computers share a copy of an infected program, every file accessible from every one of those machines can be infected. Methods of sharing include making and distributing copies of software, accessing bulletin board systems, public file servers, and obtaining source files from remote hosts using a network or electronic mail.

The probability of any new program containing malicious logic depends on the integrity of the author (or authors), the security and integrity of the computer on which they worked, on which the distribution was prepared, and on the method of distribution. Programs sent through electronic mail or posted to bulletin boards may be altered in transit, either by someone modifying them while they sit on an intermediate node, or even while they are crossing networks [131]. Further, electronic messages can easily be forged [113][130], so it is unwise to rely on such a program's stated origin.

For example, in the early 1980s a program posted to the USENET news network, an international bulletin board system, contained a command to delete all files on the system in which it was run. Some system administrators executed the program with unlimited privileges, thereby damaging their systems. Although vendors usually take care that their software contains no malicious logic, in another case a company selling software for the Macintosh[1] unwittingly delivered copies of programs infected by a computer virus which printed a message asking for universal peace [49].

6.3. Integrity of Programs

The infection phase of a virus' actions require writing to files; for reasons discussed earlier, access controls provide little protection. Typically some form of auditing is used to detect changes [14][18]; however, auditing schemes cannot prevent damage, but only attempt to provide a record of it and (possibly) indicate the culprit. The best auditing methods use a mechanism that records changes to files or their characteristics. Such schemes require kernel modifications and should be designed into new systems [54][75][93]; they typically must be added to existing systems [99]. The audit logs must also be protected from illicit modification; again, an element of trust in the underlying subsystem is needed.

Many sites simply have licenses for object code and so cannot add the required mechanisms to their kernel. These sites must scan the file system either periodically or randomly [13]. A com-

1. Macintosh is a Registered Trademark of Apple Computer

puter virus can defeat either scheme by infecting a file and restoring the uninfected version between the times the audit mechanism audited the file; the 4096 (personal computer) virus uses this technique to evade detection [85].

No program can determine if an arbitrary virus has infected a file because of the undecidability results cited earlier; however, *virus detectors* or *anti-virus agents* can check files for specific virus. Since viruses can mutate, if a virus detector reports that no infection is present, the file may contain a virus unknown to the detector or the detector may be corrupt. Maliciously altered or infected virus detectors can cause severe problems. In February 1989, at Dartmouth College, a user ran the virus detection program Interferon twice on a Macintosh with a hard disk; the first time, Interferon became infected, and the second time, it infected files on the disk. More widely known is the Trojan horse in a doctored copy of FLUSHOT [61], which caused the author of that program to rename it FSP+ to avoid confusion with the tampered version [7].

6.4. Backups and Recovery

Using backups to replace infected files, or files which contain malicious logic, may remove such programs from the system; on the other hand, if the programs on the backup medium contain malicious logic, the restoring does little good. Furthermore, as most systems make backup copies of files which have changed since the time the previous backup was made, it is in fact quite likely that files from several backups previously will need to be examined to find an uncontaminated version of the infected program.

Other vulnerabilities include corrupt backup and restore programs; if those programs contain malicious logic preventing uncorrupted software from being restored, then the backups are useless until a way is found to replace (or fix) the restore program. Similarly, unless all malicious programs are found and restored at the same time, the restoration of some uncorrupted programs may do little (for example, computer viruses still resident on the system could infect the newly-restored programs). It may in fact cause harm – some research and development systems (such as variants of the UNIX operating system) do not allow users to "lock" devices, so one user can access media mounted by another user. Thus, between the mounting and the attempt to restore, another program containing malicious logic could easily infect or erase a mounted backup.

6.5. The Human Factor

It has been said that computer viruses are a management issue, because they are introduced

by people [35]; the same may be said for all malicious logic, and computer security in general. Ideally, security procedures should balance the security and safety of the system and data with the needs of the users and systems personnel to get work done. All too often, users (and systems personnel) see them as burdens to be evaded. Lack of awareness of the reasons for security procedures and mechanisms leads to carelessness or negligence, which can in turn lead to system compromise; [98] describes an incident where an attacker obtained a password by calling the computer operator and claiming to be someone else.

Little if anything can be done to prevent compromise by trusted personnel; malicious users and system administrators can often circumvent security policy restrictions without being stopped, or even detected, by the mechanisms enforcing the policies; recall the exceptions to the security and integrity models mentioned in section 3. (See [96] for examples of these "inside jobs.") The study of computing ethics, or of a code of ethical conduct, should reduce this threat by making clear what actions are considered acceptable; should a breach occur, legal remedies may be avaliable [52][108].

6.6. Multiple Levels of Privilege

Multi-user computer systems often provide many different levels of privilege; for example, UNIX provides a separate set of privileges for each user, and one all-powerful *superuser*. Since malicious logic requires privileges to access files (either for reading or for writing), enforcing the *principle of least privilege* [107] can limit those programs.

If someone using a privileged account accidentally executes a program containing a computer virus, the virus will spread throughout the system rapidly [43]. Hence, simply logging in as a privileged user and remaining so empowered increases the possibility of accidentally triggering some form of malicious logic. More subtle is the use of programs which can cross protection domain boundaries; when the boundary being crossed involves the addition of a privilege or capability that enables the user to affect objects in many other protection domains (such as changing from an unprivileged to a privileged mode), a malicious program could wreak havoc. In general, computer systems do not force such programs to function with as few privileges as possible (the setuid and setgid mechanism of UNIX [12][20][80] violate this rule).

A related but widely-ignored problem is the use of "smart" terminals to access privileged accounts. These terminals will respond to control sequences from a host by transmitting portions of the text on their screen back to the host [50], and often perform simple editing functions for the

host. Such a terminal can issue a computer virus' commands in the name of the terminal's user when appropriate text and control sequences are sent to it (for example, by using an inter-terminal communications program or displaying files with appropriate characters in it.) These commands could instruct the computer to execute an infected program, which would run in the protection domain of the user of the terminal (and not that of the attacker). As many computers use such terminals as their consoles, and allow access to the most privileged accounts only when the user is at the console, the danger is obvious.

6.7. Direct Device Access

This is a form of the *principle of complete mediation* [107] which requires checking every access. Although multiuser systems have virtual memory protection to prevent processes from writing into each other's memory, some represent devices and memory as addressable objects (such as files). If these objects are improperly or inadequately protected, a process could bypass the virtual memory controls and write to any location in memory by placing data and addresses on the bus, thereby altering the instructions and data in another's memory space (the "core war" games [40] are examples of programs that did this). Under similar conditions any process could write to disks without the kernel's intervention, anyone can change executable programs regardless of their protection – and a virus can easily spread by taking advantage of the (lack of) protection.

7. Conclusion

This paper has described the threats that computer viruses pose to research and development multiuser computer systems; it has attempted to tie those programs with other, usually simpler, programs that can have equally devastating effects. Although reports of malicious programs in general abound, no non-experimental computer viruses have been reported on mainframe systems.[1] Noting that the number of people with access to mainframes is relatively small compared to the number with access to personal computers [127], Highland suggests [61] that as malicious people make up a very small fraction of all computer programmers, most likely fewer malicious people use research and development systems than personal computers. A more plausible argument, advanced by Fåk [47] and supported by Kurzban [76] is that, as only programmers can create com-

1. Cohen tantalizingly claims that at least one has been found, but reports no other details [26]. Suppression of details (or, more commonly, the existence) of attacks, virus or otherwise, is common; it is estimated that victims report only 10% to 35% of computer crimes in general [116][126], in part to prevent embarrassment or loss of public confidence in the company, or to avoid the expense of gathering sufficient evidence to prosecute the offender [98].

puter viruses, and malicious mainframe programmers can accomplish their goals with less trouble than writing a computer virus, computer virus attacks will most likely be confined to personal computers. However, this overlooks attacks motivated by a perceived intellectual challenge of creating a virus, by a desire to demonstrate limits of existing security mechanisms, or attacks launched simply by carelessness or error [95].[1] How influential these latter factors are is not yet known.

Should an attacker use a computer virus or other malicious program, security mechanisms currently in use will probably prove ineffective. As with malicious programs in general, though, people can prepare for such an attack and minimize the damage done. This paper has described several vulnerabilities in the research and development environment that malicious programs could exploit, and also discussed research underway to improve defenses against malicious logic. How effective these new mechanisms will be in reducing the vulnerabilities, only time will tell.

Acknowledgments: Thanks to Holly Bishop, André Bondi, Emily Bryant, Peter Denning, Donald Johnson, John Rushby, Ken Van Wyk, and the anonymous referees, all of whose comments and advice improved the quality of the paper greatly. Josh Alden of the Dartmouth Virus Clinic described the Interferon infection incident, Robert Van Cleef and Gene Spafford helped reconstruct the USENET logic bomb incident, and Ken Thompson confirmed that he had indeed doctored an *internal* version of the C compiler as described in [125]. My thanks to them also.

References

[1] G. Al-Dossary, "Computer Virus Prevention and Containment on Mainframes," *Computers and Security* **9**(2) (Apr. 1990) pp. 131-137.

[2] L. Adelman, "An Abstract Theory of Computer Viruses,", *Advances in Cryptology – CRYPTO '88 Proceedings*, Springer-Verlag, New York, NY (Aug. 1988) pp. 354-374.

[3] M. Adkins, G. Dolsen, J. Heaney, and J. Page, "The Argus Security Model," *Twelfth National Computer Security Conference Proceedings* (Oct. 1989) pp. 123-134.

[4] J. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom Air Force Base, MA (1974).

[5] A. Avižienis, "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactios on Software Engineering* **SE-11**(12) (Dec. 1985) pp. 1491-1501.

1. It is worth noting that the author of the Internet worm stated that the worm disabled machines due to a programming error [90].

[6] A. Avižienis, M. Lyu, and W. Schutz, "In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Control Software," Technical Report CSD-870060, University of California, Los Angeles, CA (Nov. 1987).

[7] D. Bader, "Bad Versions of FLUSHOT (for IBM PC)," *Virus-L Digest* **1**(8) (Nov. 15, 1988).

[8] D. Bauer and M. Koblentz, "NDIX – A Real-Time Intrusion Detection Expert System," *1989 Summer USENIX Conference Proceedings* (June 1988) pp. 261-274.

[9] D. Bell and L. LaPadula, "Secure Computer Systems: Unified Exposition and MULTICS Interpretation," Technical Report MTR-2997, MITRE Corporation, Bedford, MA (July 1975).

[10] B. Bershad and C. Pinkerton, "Watchdogs: Extending the UNIX File System," *1988 Winter USENIX Conference Proceedings* (Feb. 1988) pp. 267-276.

[11] K. Biba, "Integrity Considerations for Secure Computer Systems," Technical Report ESD-TR-76-372, Air Force Electronic Systems Division, Hanscom Air Force Base, MA (1977).

[12] M. Bishop, "How to Write a Setuid Program," *;login:* **12**(1) (Jan. 1987) pp. 5-11.

[13] M. Bishop, "Auditing Files on a Network of UNIX Machines," *Proceedings of the UNIX Security Workshop* (Aug. 1988) pp. 51-52.

[14] M. Bishop, "A Model of Security Monitoring," *Proceedings of the Fifth Annual Computer Security Applications Conference* (Dec. 1989) pp. 46-52.

[15] W. Boebert and C. Ferguson, "A Partial Solution to the Discretionary Trojan Horse Problem," *Proceedings of the Eighth Computer Security Conference* (sep. 1985) pp. 245-253.

[16] W. Boebert and R. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *Proceedings of the Eighth Computer Security Conference* (Sep. 1985) pp. 18-27.

[17] W. Boebert, W. Young, R. Kain, and S. Hansohn, "Secure Ada Target: Issues, System Design, and Verification," *Proceedings of the 1985 Symposium on Security and Privacy* (Apr. 1985) pp. 176-183.

[18] D. Bonyun, "The Role of a Well Defined Auditing Process in the Enforcement of Privacy Policy and Data Security," *Proceedings of the 1981 Symposium on Security and Privacy* (Apr. 1981) pp. 19-25.

[19] J. Brunner, *The Shockwave Rider*, Ballanw York City, NY (1975).

[20] S. Bunch, "The Setuid Feature in UNIX and Security," *Tenth National Comnputer Security*

Conference Proceedings (Sep. 1987) pp. 245-253.

[21] R. Burger, *Computer Viruses – A High-Tech Disease*, Abacus, Grand Rapids, MI (1988).

[22] L. Chen, "Improving Software Reliability by N-Version Programming," Technical Report Eng-7843, University of California, Los Angeles, CA (Aug. 1978).

[23] D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 Symposium on Security and Privacy* (Apr. 1987) pp. 184-194.

[24] F. Cohen, "Computer Viruses: Theory and Experiments," *Seventh DOD/NBS Computer Security Conference Proceedings* (Sep. 1984) pp. 240-263.

[25] F. Cohen, "Computer Viruses: Theory and Experiments," *Computers and Security* **6**(1) (Feb. 1987) pp. 22-35.

[26] F. Cohen, "On the Implications of Computer Viruses and Methods of Defense," *Comput* **7**(2) (Apr. 1988) pp. 167-184.

[27] F. Cohen, "Maintaining a Poor Person's Information Integrity," *Computers and Security* **7**(5) (Oct. 1988) pp. 489-494.

[28] F. Cohen, "Practical Defenses Against Computer Viruses," *Computers and Security* **8**(2) (Apr. 1989) pp. 149-160.

[29] F. Cohen, "Computational Aspects of Computer Viruses," *Computers and Security* **8**(4) (June 1989) pp. 325-344.

[30] F. Cohen, *A Short Course on Computer Viruses*, ASP Press, Pittsburgh, PA (1990).

[31] S. Crocker and M. Pozzo, "A Proposal for a Verification-Based Virus Filter," *Proceedings of the 1989 IEEE Symposium on Security and Privacy* (May 1989) pp. 319-324.

[32] J. David, "Treating Viral Fever" *Computers and Security* **7**(2) (Apr. 1988) pp. 255-258.

[33] G. Davida, Y. Desmedt, and B. Matt, "Defending Systems Against Viruses through Cryptographic Authentication," *Proceedings of the 1989 Symposium on Security and Privacy* (May 1989) pp. 312-318.

[34] G. Davida and B. Matt, "UNIX Guardians: Delegating Security to the User," *Proceedings of the UNIX Security Workshop* (Aug. 1988) pp. 14-23.

[35] H. DeMaio, "Viruses – Management Issue," *Computers and Security* **8**(5) (Oct. 1989) pp.

381-388.

[36] D. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering* **SE-13**(2) (Feb. 1987) pp. 222-232.

[37] P. Denning, "The Science of Computing: Computer Viruses," *American Scientist* **76**(3) (May 1988) pp. 236-238.

[38] P. Denning, *Computers Under Attack: Intruders, Worms, and Viruses*, Addison-Wesley Publishing Co., Reading, MA (1990),

[39] A. Dewdeney, "Computer Recreations: A Core War Bestiary of Viruses, Worms, and Other Threats to Computer Memories," *Scientific American* **252**(3) (Mar. 1985) pp. 14-23.

[40] A. Dewdeny, "Computer Recreations," *Scientific American* **256**(1) (Jan. 1987) pp. 14-20.

[41] *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, Department of Defense (Dec. 1985).

[42] D. Downs, J. Rub, K. Kung, and C. Jordan, "Issues in Discretionary Access Control," *Proceedings of the 1984 IEEE Symposium on Security and Privacy* (Apr. 1984) pp. 208-218.

[43] T. Duff, "Experiences with Viruses on UNIX Systems," *Computing Systems* **2**(2) (Spring 1989) pp. 155-172.

[44] M. Eichin and J. Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," *Proceedings of the 1989 IEEE Symposium on Security and Privacy* (Apr. 1989) pp. 326-343.

[45] T. Eisenberg, D. Gries, J. Hartmanis, D. Holcomb, M. Lynn, and T. Santoro, *The Computer Worm: A Report to the Provost of Cornell University on an Investigation Conducted by the Commission of Preliminary Enquiry*, Cornell University, Ithaca, NY (Feb. 1989).

[46] P. Elmer-DeWitt, "Invasion of the Data Snatchers: A Virus Epidemic Strikes Terror in the Computer World," *Time* (Sep. 26, 1988) pp. 62-67.

[47] V. Fåk, "Are We Vulnerable to a Virus Attack: A Report from Sweden," *Computers and Security* **7**(2) (Apr. 1988) pp. 151-155.

[48] R. Farrow, *UNIX System Security*, Addison-Wesley Publishing Co., Reading, MA (1991).

[49] P. Fites, P. Johnston, and M. Kratz, *The Computer Virus Crisis*, Van Nostrand Reinhold, New York City, NY (1988).

[50] M. Gabriele, ""Smart" Terminals for Trusted Computer Sysztems," *Ninth National Computer Security Conference Proceedings* (Sep. 1986) pp. 16-20.

[51] P. Garnett, "Selective Disassembly: A First Step Towards Developing a Virus Filter," *Fourth Aerospace Computer Security Conference* (Dec. 1988) pp. 2-6.

[52] M. Gemignani, "Viruses and Criminal Law," *CACM* **32**(6) (June 1989) pp. 669-671.

[53] W. Gleissner, "A Mathematical Theory for the Spread of Computer Viruses," *Computers and Security* **8**(1) (Feb. 1989) pp. 35-41.

[54] V. Gligor, C. Chandersekaran, R. Chapman, L. Dotterer, M. Hecht, W. Jiang, A. Johri, G. Luckenbaugh, and N. Vasudevan, "Design and Implementation of Secure Xenix," *IEEE Transactions on Software Engineering* **SE-13**(2) (Feb. 1987) pp. 208-220.

[55] F. Grampp and R. Morris, "UNIX Operating System Security," *AT&T Bell Laboratories Technical Journal* **63**(8) (Oct. 1984) pp. 1649-1672.

[56] J. Green and P. Sisson, "The "Father Christmas" Worm," *Twelfth National Computer Security Conference Proceedings* (Oct. 1989)pp. 359-368.

[57] K. Hafner, "Is Your Computer Secure?," *Business Week* (Aug. 1, 1987) pp. 64-72.

[58] J. Haigh and W. Young, "Extending the Non-Interference Version of MLS for SAT," *Proceedings of the 1986 IEEE Symposium on Security and Privacy* (Apr. 1986) pp. 232-239.

[59] H. Highland, "Random Bits and Bytes: Case History of a Virus Attack," *Computers and Security* **7**(1) (Feb. 1988) pp. 3-5.

[60] H. Highland, "Random Bits and Bytes: Case History of a Virus Attack," *Computers and Security* **7**(1) (Feb. 1988) pp. 6-7.

[61] H. Highland, "Random Bits and Bytes: Computer Viruses – A Post-Mortem," *Computers and Security* **7**(2) (Apr. 1988) pp. 117-127.

[62] H., Highland, "The Brain Virus: Fact and Fantasy," *Computers and Security* **7**(4) (Aug. 1988) pp. 367-370.

[63] H. Highland, "Random Bits and Bytes: Another Poor Password Disaster," *Computers and Security* **9**(1) (Feb. 1990) p. 10.

[64] L. Hoffman, *Rogue Programs: Viruses, Worms, and Trojan Horses*, Van Nostrand Reinhold, New York City, NY (1990).

[65] Homer, *The Odyssey,* Penguin Books, New York City, NY (1946).

[66] H. Israel, "Computer Viruses: Myth or Reality?," *Tenth National Computer Security Conference Proceedings* (Sep. 1987) pp. 226-230.

[67] M. Joseph, "Towards the Elimination of the Effects of Malicious Logic: Fault Tolerance Approaches," *Tenth National Computer Security Conference Proceedings* (Sep. 1987) pp. 238-244.

[68] M. Joseph and A. Avižienis, "A Fault Tolerant Approach to Computer Viruses," *Proceedings of the 1988 Symposium on Security and Privacy* (Apr. 1988) pp. 52-58.

[69] J. Juni and R. Ponto, "Computer-Virus Infection of a Medical Diagnostic Computer," *New England Journal of Medicene* **320**(12) (Mar. 12, 1989) pp. 811-812.

[70] P. Karger, "Limiting the Damage Potential of Discretionary Trojan Horses," *Proceedings of the 1987 Symposium on Security and Privacy* (Apr. 1987) pp. 32-37.

[71] P. Karger and R. Schell, "MULTICS Security Evaluation: Vulnerability Analysis," Technical Report ESD-TR-74-193, Air Force Electronic Systems Division, Hanscom Air Force Base, MA (1974).

[72] B. Kernighan and T. Plauger, *The Elements of Programming Style*, McGraw-Hill Book Co., New York City, NY (1974).

[73] J. Knight and N. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-version Programming," *IEEE Transactions on Software Engineering* SE-**12**(1) (Jan. 1986) pp. 96-109.

[74] J. Knight and N. Leveson, "On N-version Programming," *Software Engineering Notes* **15**(1) (Jan. 1990) pp. 24-35.

[75] S. Kramer, "Linus IV – An Experiment in Computer Security," *Proceedings of the 1984 Symposium on Security and Privacy* (Apr. 1984) pp. 24-31.

[76] S. Kurzban, "Viruses and Worms -- What Can You Do?," *SIGSAC Review* **7**(1) pp. 16-32.

[77] N. Lai and T. Gray, "Strengthening Discretionary Access Controls to Inhibit Trojan Horses and Computer Viruses," *1988 Summer USENIX Conference Proceedings* (June 1988) pp. 275-286.

[78] B. Lampson, "Hints for Computer System Design," *IEEE Software* **1**(1) (Jan. 1984) pp. 11-28.

[79] R. Levin, *Computer Virus Handbook*, McGraw-Hill Book Co., New York City, NY (1990).

[80] T. Levin, S. Padilla, and C. Irvine, "A Formal Model for UNIX Setuid," *Proceedings of the 1989 Symposium on Security and Privacy* (May 1989) pp. 73-83.

[81] P. Lewis, "The Executive Computer: A Virus Carries Fatal Complications," *New York Times* (June 26, 1988) p. C-11.

[82] J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers*, RFC-1115 (Aug. 1989).

[83] S. Lipner, "Non-Discretionary Controls for Commercial Applications," *Proceedings of the 1982 Symposium on Security and Privacy* (Apr. 1982) pp. 2-10.

[84] T. Lunt and R. Jagannathan, "A Prototype Real-Time Intrusion-Detection Expert System," *Proceedings of the 1988 Symposium on Security and Privacy* (Apr. 1988) pp. 59-66.

[85] J. McAfee, "4096 and 1260 Viruses (PC)," *Virus-L Digest* 3(27) (Jan. 31, 1990), submitted by A. Roberts.

[86] J. McAfee and C. Haynes, *Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System*, St. Martin's Press, New York City, NY (1989).

[87] M. Machtey and P. Young, *An Introduction to the General Theory of Algorithms*, Elsevier North Holland, Inc., New York City, NY (1978).

[88] J. Markoff, "'Virus' in Military Computers Disrupts Systems Nationwide," *New York Times* (Nov. 4, 1988) p. A-1.

[89] J. Markoff, "Top-Secret, And Vulnerable," *New York Times* (Apr. 25, 1988) p. A-1.

[90] J. Markoff, "Student Says Error in Experiment Jammed a Network of Computers," *New York Times* (Jan. 19, 1990) p. A-19.

[91] V. McLellan, "Computer Systems Under Seige," *New York Times* (Jan. 31, 1989) p. C-3.

[92] R. Merkle, "A Fast Software One Way Hash Function," *unpublished.*

[93] G. Miller, S. Sutton, M. Matthews, J. Yip, and T. Thomas, "Integrity Mechanisms in a Secure UNIX: GOULD UTX/32S," *AIAA/ASIS/DODCI Second Aerospace Computer Security Conference: A Collection of Technical Papers* (Dec. 1986) pp. 19-26.

[94] W. Murray, "The Application of Epidemiology to Computer Viruses," *Computers and Security* 7(1) (Feb. 1988) pp. 139-150.

[95] P. Neumann and D. Parker, "A Summary of Computer Misuse Techniques," *Twelfth National Computer Security Conference Proceedings* (Oct. 1989) pp. 396-407.

[96] A. Norman, *Computer Insecurity*, Chapman and Hall, New York City, NY (1983).

[97] J. Page, "An Assured Pipeline Integrity Scheme for Virus Protection," *Twelfth National Computer Security Conference Proceedings* (Oct. 1989) pp. 369-377.

[98] D. Parker, *Crime by Computer*, Charles Scribner's Sons, New Yok City, NY (1976).

[99] J. Picciotto, "The Design of an Effective Auditing Subsystem," *Proceedings of the 1987 Symposium on Security and Privacy* (Apr. 1987) pp. 13-22.

[100] G. Popek and B. Walker, *The LOCUS Distributed System Architecture*, The MIT Press, Cambridge, MA (1985).

[101] M. Pozzo and T. Gray, "A Model for the Containment of Computer Viruses," *AIAA/ASIS/DODCI Second Aerospace Computer Security Conference* (Dec. 1986) pp. 11-18.

[102] M. Pozzo and T. Gray, "An Approach to Containing Computer Viruses," *Computers and Security* **6**(4) (Aug. 1987) pp. 321-331.

[103] B. Randell, P. Lee, and P. Treleaven, "Reliability Issues in Computing System Design," *Computing Surveys* **10**(2) (June 1978) pp. 167-196.

[104] D. Ritchie, "Joy of Reproduction," USENET newsgroup *net.lang.c* (Nov. 4, 1982).

[105] R. Roberts, *Computer Viruses*, Compute! Books, Greensboro, NC (1988).

[106] J. Rochlis and M. Eichin, "With Microscope and Tweezers: The Worm from MIT's Perspective," *CACM* **32**(6) (June 1989) pp. 689-698.

[107] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE* **63**(9) (Sep. 1975) pp. 1278-1308.

[108] P. Samuelson, "Can Hackers Be Sued for Damages Caused by Computer Viruses?," *CACM* **32**(6) (June 1989) pp. 666-669.

[109] O. Saydjari, J. Beckman, and J. Leaman, "Locking Computers Securely," *Tenth National Computer Security Conference Proceedings* (Sep. 1987) pp. 129-141.

[110] O. Saydjari, J. Beckman, and J. Leaman, "LOCK Trek: Navigating Uncharted Space," *Proceedings of the 1989 Symposium on Security and Privacy* (May 1989) pp. 167-175.

[111] R. Schatz, "New 'Virus' Infects NASA Macintoshes," *Washington Post* (Apr. 18, 1988),

sec. Washington Business, p. 25.

[112] J. Schoch and J. Hupp, "The "Worm" Programs – Early Experiences with a Distributed Computation," *CACM* **25**(3) (Mar. 1982) pp. 172-180.

[113] P. Scott, "Re: Faking Internet Mail [Re: RISKS-8.27]," *Forum on the Risks to the Public in Computers and Related Systems* **8**(28) (Feb. 19, 1989).

[114] D. Seeley, "Password Cracking: A Game of Wits," *CACM* **32**(6) (June 1989) pp. 700-703.

[115] D. Seeley, "A Tour of the Worm," *Proceedings of USENIX Winter '89* (Jan. 1989) pp. 287-304.

[116] P. Singer, "Trying to Put a Brake on Computer Theft," *New York Times* (Mar. 2, 1986) p. WC-17.

[117] K. Smith, "Tales of the Damned," *UNIX Review* **6**(2) (Feb. 1988) pp. 45-50.

[118] T. Smith, "User Definable Domains as a Mechanism for Implementing the Least Privilege Principle," *Ninth National Computer Security Conference Proceedings* (Sep. 1986) pp. 143-148.

[119] E. Spafford, K. Heaphy, and D. Ferbrache, *Computer Viruses: Dealing with Electronic Vandalism and Programmed Threats*, ADAPSO, Arlington, VA (1989).

[120] E. Spafford, "Crisis and Aftermath," *CACM* **32**(6) (June 1989) pp. 678-687.

[121] E. Spafford, "The Internet Worm Program: An Analysis," *ACM Computer Communications Review* **19**(1) (Jan. 1989).

[122] E. Spafford, K. Heaphy, and D. Ferbrache, *Computer Viruses: Dealing with Electronic Vandalism and Programmed Threats*, ADAPSO, Arlington, VA (1989).

[123] C. Stoll, "An Epidemiology of Viruses & Network Worms," *Twelfth National Computer Security Conference Proceedings* (Oct. 1989)pp. 369-377.

[124] H. Teng, K. Chen, and S. Lu, "Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns," *Proceedings of the 1990 Symposium on Research in Security and Privacy* (May 1990) pp. 278-284.

[125] K. Thompson, "Reflections on Trusting Trust," *Communications of the ACM* **27**(8) (Aug. 1984) pp. 761-763.

[126] United States Comptroller General, "Computer-Related Crimes in Federal Programs," Re-

port FGMSD-76-27, United States Government Printing Office, Washington, D. C. (Apr. 27, 1976).

[127] United States Congress Office of Technology Assessment, *Defending Secrets, Sharing Data: New Locks and Keys for Electronic Information*, Report OTA-CIT-310, United States Government Printing Office, Washington, D. C. (Oct. 1987).

[128] Virgil, *The Æneid*, Random House, New York City, NY (1983).

[129] C. von Rospach, "How to Post a Fake," *Forum on the Risks to the Public in Computers and Related Systems* 4(75) (Apr. 20, 1987).

[130] V. Voydock and S. Kent, "Security Mechanisms in High-Level Network Protocols," *Computing Surveys* 15(2) (June 1983) pp. 135-171.

[131] S. Wiseman, "Preventing Viruses in Computer Systems," *Computers and Security* 8(5) (Aug. 1989) pp. 427-432.

[132] I. Witten, "Computer (in)security: Infiltrating Open Systems," *Abacus* 4(4) (1987) pp. 7-25.

[133] C. Young, "Taxonomy of Computer Virus Defense Mechanisms," *Tenth National Computer Security Conference Proceedings* (Sep. 1987) pp. 220-225.

Sidebar 1 – The First Trojan Horse

There are many contradictory versions of this story; it appears only briefly in The Odyssey *([65], Book VIII), but later writers elaborated it considerably. Aeneas, a Trojan survivor of the sacking of the city, tells the following version to Queen Dido of Carthage during his wanderings that ended with the founding of Rome ([129], Book II).*

After many years of besieging Troy and failing to take the city, the Greeks, on the advice of Athene, built a large wooden horse in which many Greek soldiers hid. The horse was inscribed with a prayer to Athene to grant the Greeks safe passage home, and then the Greek army pretended to sail home.

The next morning, the Trojans discovered the siege had been lifted and went to examine the wooden horse. One of the elders, Thymoetes, noticed the inscription, and urged the horse be brought into the city and placed in Athene's temple. Others counseled that the horse must be destroyed; in particular Laocoon, a priest of Apollo, emphatically threw a spear against the horse's belly as he cried that he did not trust Greeks bearing gifts.

Meanwhile, shepherds allied with the Trojans brought over a Greek soldier named Sinon. Sinon explained that the Greeks had desecrated Apollo's shrine and killed a virgin attendant in a raid, so to appease Apollo they had to sacrifice one of their men. Sinon was chosen, and he promptly fled and was abandoned when the Greeks left for home. Under further questioning, the captive claimed that one night Odysseus and Diomede desecrated Athene's shrine, turning their protecting goddess against them. Calchas, the Greeks' priest, advised that the horse must be built to appease the goddess before they could leave; and the horse was made big so the Trojans could not get it into their city, for if they did their triumph over the Greeks would be assured.

At this moment, two sea serpents slithered out of the waters and crushed Laocoon and his sons to death. Believing this to be retribution for his profaning an offering to Athene, the Trojans immediately breached the walls of the city and pulled the horse inside.

That night, as the Trojans celebrated, they did not notice Sinon slip out to the horse and open a trap door through which the Greek soldiers emerged, nor did they see the Greeks opening the gates to the city. The Greek forces had by this time returned, and they sacked the city. Aeneas and his companions alone escaped.

Sidebar 2 – Anatomy of a Virus

This pseudocode fragment shows how a virus might be coded. It is quite simplistic, and considerable elaboration is possible, but all viruses follow this structure in some form or another.

```
beginvirus:
     if spread-condition then begin
          for some set of target files do begin
               if target is not infected then begin
                    determine where to place virus instructions
                    copy instructions from beginvirus
                         to endvirus into target
                    alter target to execute added instructions
               end;
          end;
     end;
     perform some action
     goto beginning of infected program
endvirus:
```

First, the virus determines if it is to spread; if so, it locates a set of target files it is to infect,

and copies itself into a convenient location within the target file. It then alters portions of the target to ensure the inserted code will be executed at some time. For example, the virus may append itself just beyond the end of the instruction space and then adjust the entry points used by the loader so that the added instructions will execute when the target program is next run. This is the *infection phase* It then performs some other action (the *execution phase* Note that the execution phase can be null and the instructions still constitute a virus; but if the infection phase is missing, the instructions are *not* a virus. Finally, it returns control to the program currently being run.

The Lehigh virus [59] worked this way. The *spread-condition* was that "there is an uninfected boot file on the disk," the *set of target files* was "the uninfected boot file," and *perform some action* was to increment a counter and test to see if the counter had reached 4; if so, it would erase the disk.

Sidebar 3 – Some Suggested Guidelines for UNIX-based Systems

This list of suggestions, intended for a basic, "vanilla" UNIX-based computer system, will help prevent the introduction of malicious logic, like computer viruses, into the computer system, and also lessen the chances of accidentally invoking programs with that type of logic. Sophisticated attackers can render these methods ineffective because the weaknesses they seek to patch are fundamental to the design and use of the computer system, and anything effective would require changing the system more than is practical. Still, following these suggestions may help.

1. Set the environment variables (such as **PATH**) to access trusted programs before accessing untrusted programs of the same name.

   The UNIX shell checks the value of the variable **PATH** for a list of directories to check for programs. In the example in §6.1., the system administrator had put the current working directory before the system directories; hence the user's program, not the system one, was executed.

2. Do not execute a program obtained from an untrusted source without checking the source code thoroughly.

   This rule presumes that the underlying computing base (compiler, loader, operating system, etc.) are all uncorrupted; if this assumption is false, malicious logic may be inserted during compilation, linking, or execution. An obvious corollary is to test all such software in an environment with very limited privileges before installing it, and *never* to test the program where it can access critical or irreplaceable files, or as a highly-privileged user.

3. Design and implement some auditing scheme to ensure that files' access control permissions match the settings specified in an access control plan.

   This requires first, that some security policy designating who has access to what files and how be created; and second, that some enforcement mechanism be implemented. Note the *caveat*: if the audit log created by that mechanism, or the mechanism itself, can be tampered with, the introduction of malicious logic into the system can be done undetectably. However, depending on the security mechanisms implementing the auditing and the access to the log, this may require some sophistication. (Or, it may not.)

4. Check the integrity of system files to ensure they have not changed unexpectedly.

   This is really a corollary to the previous rule. Note that the checksums computed at installation must be protected, since an attacker could change a file, then compute its new checksum and replace the stored checksum with it. Again, this requires that the underlying system be trusted to provide such protection to the checksum program, the stored checksums, and the audit program comparing the two.

5. Backups should be made regularly and kept as long as reasonable.

   Typically, sites make both daily and weekly incremental backups (which save all files that have changed since the last incremental backup of the same period); then once a month they simply make a copy of all file systems. Enough of each kind is saved to be able to restore the system to its current state. Notice that if restoring to eliminate a malicious program, the restored version of the program should also be *thoroughly* checked.

6. Discuss with your systems staff and users the reasons for, and effects of, any actions taken for security reasons.

   The system staff should cultivate good relations with the users and vendors, should be certain to explain the reasons for all security policies, and should assist users whenever possible in providing a pleasant and secure working environment, acting as an intermediary between them and the vendors if need be. Users and staff should know what constitutes a breach of security, and there should be a well-designed set of procedures for handling breaches. Thinking through the best procedures for a particular installation carefully, putting them into place tactfully, and explaining them fully, will do far more to prevent security problems than any quick action.

7. All installations should keep the original distribution of the computer system in a safe place, and make and protect backups as well.

   If malicious programs are determined to be rampant on the system, the administrators should reload the original compilation and installation software from the distribution medium and recompile and regenerate all system files after checking all sources thoroughly. This assumes that the (distributed) compilation and installation software is not infected and the program loading that software does not infect it. As always, the elements of trust are present here.

8. When reading backups, mount the backup medium in such a way that it cannot be changed or erased.

   The reason is explained in the text. Note this means preventing modification access *by the hardware*, for example by removing the write ring from a tape. If the prevention mechanism is done in software, it can be infected and/or disabled by a malicious program. Here, the element of trust is in the hardware mechanism working correctly.

9. Access privileged accounts only when necessary, and then for as brief a time as possible.

   Should someone using a privileged account accidentally execute a program containing a computer virus, the virus will spread throughout the system rapidly. This is less likely to happen if those accounts are used only when necessary; even so, a window of vulnerability still exists. Computers designed with security in mind typically limit the power of privileged accounts, in some cases very drastically.

10. Write as few privileged programs as possible.

    The more programs that can cross protection domain boundaries while executing, the more potential targets for the addition of malicious logic exist. This suggestion essentially recommends minimizing the number of programs that can be modified to provide an attacker with entry to the privileged state.

11k. Never use a smart terminal to access a privileged account.

111. If using a smart terminal to access a privileged account, never allow an inter-terminal communications program to write to the terminal, never read electronic mail from that terminal, and do not look at files the contents of which are unknown or suspect.

    Note that the second version is much weaker, because a malicious program could tamper with an executable program and cause it to display the control sequences to produce the requisite commands from the terminal. The privileged user executing such a command springs the trap. Any file the malicious program could write to can be similarly booby-trapped.

13. Prevent users from accessing devices and memory directly.

    If memory and devices are objects addressable by the user, the access control plan described earlier should include these objects and prevent direct access to them. Specifically, the device and memory files on UNIX systems should *never* have any *world* permissions set; this gives users direct access to memory and to the raw device, and allows them to bypass the UNIX access control mechanisms.

Sidebar 4 – Forums that Discuss Viruses

The VIRUS-L mailing list, originating at Lehigh University and moderated by Kenneth R. van Wyk, is a forum for discussing all aspects of computer viruses. Participants often describe computer viruses in that forum before magazines or other publications do so; they have also discussed virus remedies, protection against viruses, the theory behind viruses, and how the media handles reports of computer viruses. To subscribe, send an electronic mail message containing only the line

SUB VIRUS-L *your name*

to *LISTSERV@LEHIIBM1.BITNET*. Back issues of the digest are available by anonymous ftp from *IBM1.CC.LEHIGH.EDU* or *cert.sei.cmu.edu*; users not on the internet may use the BITNET pro-

tocol of sending to *LISTSERV@LEHIIBM1.BITNET* an electronic mail message containing only the line

GET VIRUS-L LOG*yymmx*

where *yy* is the last two digits of the year, *mm* the number of the month, and *x* a letter indicating the number of the week in the month. For example, the line

GET VIRUS-L LOG8901B

requests the digests issued in the second week of January, 1989.

A second mailing list, VALERT-L, is used to announce viruses *only* any discussion is to take place in the VIRUS-L list. To subscribe, send an electronic mail message containing only the line

SUB VALERT-L *your name*

to *LISTSERV@LEHIIBM1.BITNET*. Any message sent to VALERT-L is cross-posted to VIRUS-L when the next digest appears.

Peter Neumann of SRI International moderates the Forum on Risks to the Public in Computers and Related Systems, or RISKS, list. This mailing list focuses on the risks involved in computer technology, and has discussed implications of viruses, although with a thrust different than the VIRUS-L mailing list. To subscribe, if on the Internet, send an electronic mail message to *RISKS-request@CSL.SRI.COM*; if on BITNET, send an electronic mail message containing only the line

SUBSCRIBE MD4H *your name*

to *LISTSERV@CMUCCVMA.BITNET*, or

SUBSCRIBE RISKS *your name*

to *LISTSERV@UGA.BITNET*, *LISTSERV@UBVM.BITNET*, or *LISTSERV@FINHUTC.BITNET*. Back issues of the digest are available by anonymous ftp from *crvax.sri.com* in the directory "sys$user2:[risks]" and are named as

RISKS-*v.nn*

where *v* is the volume and *nn* the number within the volume.