Testimony before the House Armed Services Committee
Subcommittee on
Terrorism, Unconventional Threats and Capabilities

"Cyber Terrorism: The New Asymmetric Threat"

24 July 2003

Statement of
Eugene H. Spafford
Professor and Director
Purdue University Center For Education and Research in Information Assurance and Security (CERIAS)

Co-Chair of The U.S. Public Policy Committee
of The Association For Computing Machinery (USACM)

Member of the Board of Directors
of the Computing Research Association (CRA)

Table of Contents

Introduction

Thank you Chairman Saxton and Ranking Member Meehan for the opportunity to testify at this hearing. Threats from malicious software have been steadily growing over the last 15 years and currently present a substantial danger to information systems used by the U.S. military, the civilian government, industry, academia, and the general public. So many of those systems are interconnected and dependent on each other that threats to one segment often spread to all the others. Because malicious software uses victim computers to perpetuate the attack, it presents an asymmetric threat to which U.S. computer systems are particularly vulnerable. In this testimony I will present a short primer on various types of malicious software, their history, their operation and threat, and some of the defenses we can deploy. I wish to stress at the outset, however, that the threat is significant, and the major strategies being taken by government to address this threat are palliative rather than truly preventative.

By way of introduction, I am a professor of Computer Sciences at Purdue University, a professor of Philosophy (courtesy), a professor of Communication (courtesy) and the Director of the Center for Education and Research in Information Assurance and Security. CERIAS is a campus-wide multidisciplinary Center, with a mission to explore important issues related to protecting computing and information resources. We conduct advanced research in several major thrust areas, we educate students at every level, and we have an active community outreach program. CERIAS is the largest such center in the United States, and we have a series of affiliate university programs working with us in Illinois, Iowa, North Carolina, the District of Columbia, Ohio, Virginia, and New York State. CERIAS also has a close working relationship with a dozen major commercial firms and government laboratories.

In addition to my role as an academic faculty member, I also serve on several boards of technical advisors, including those of Tripwire, Arxan, Microsoft, DigitalDoors, Unisys, and Open Channel Software; and I have served as an advisor to Federal law enforcement and defense agencies, including the FBI, the Air Force and the NSA. I am currently a member of the Air Force Scientific Advisory Board, and I have been nominated for membership on the President's Information Technology Advisory Committee. I have been working in information security issues for 25 years, and working with malicious software for over 15 years.

I began this document by listing my affiliations with ACM and CRA. This testimony is not an official statement by either organization, but is consistent with their overall goals and aims. ACM is a nonprofit educational and scientific computing society of about 75,000 computer scientists, educators, and other computer professionals committed to the open interchange of information concerning computing and related disciplines. USACM, of which I serve as the co-chair, acts as the focal point for ACM's interaction with the U.S. Congress and government organizations. USACM seeks to educate and assist policy-makers on legislative and regulatory matters of concern to the computing community. The Computing Research Association is an association of more than 180 North American academic departments of computer science and computer engineering, industry and academic laboratories, and affiliated professional societies. The CRA is

particularly interested in issues that affect the conduct of computing research in the USA. Both organizations stand ready to provide expertise and advice upon request.

Definitions and History[1]

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 *fault* or *bug* — perhaps the most common cause of unexpected program behavior. If the source of the instructions was an individual who intended that some abnormal behavior occur, then we consider this malicious coding; various authorities have sometimes referred to this code as *malware* and *vandalware*. These names relate to the usual intent of such software.

There are many distinct programmed threats that are characterized by the way they behave, how they are triggered, and how they spread. Coupled with these characteristics are a number of different methods of deployment and behavior. 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 being caused by a virus. This is unfortunate, as most problems are from other causes (including, most often, operator error or coding faults). 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 analogous to a biological virus. The word *virus* itself is Latin for *poison*. Biological viral infections are spread by the virus (a small shell containing genetic material) inserting its contents into a far larger host cell. The cell then is infected and converted into a biological factory producing replicants of the virus.

Similarly, a computer virus is typically a segment of computer code or a macro 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 a "program" is more than a simple application: startup code, word processing document macros, spreadsheets, and window systems also can be infected.

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

[1] Portions of this text are derived from my article *Virus* in Internet Beseiged: Countering Cyberspace Scofflaws; Dorothy and Peter Denning, eds.; Addison-Wesley, 1997.

The first use of the term *virus* to refer to unwanted computer code was by Gregory Benford. As related by Dr. Benford in correspondence with me[2] , he published the idea of a virus in 1970 in the May issue of Venture Magazine. His article specifically termed the idea "computer virus" and described a program named *Virus* — and tied this to the sale of a program named *Vaccine* to defeat it. All this came from his experience as a programmer and research physicist at the (then) Lawrence Radiation Lab in Livermore. He and the other scientists noticed that "bad code" could self-reproduce among lab computers, and eventually get onto the ARPANet. He tried writing and launching some "viruses" and they succeeded with surprising ease. Professor Benford's friend, the science fiction author David Gerrold, later incorporated this idea into a series of short stories in the early 1970s that were later merged into a novel in 1972: When Harlie Was One.

Fred Cohen more formally defined the term *computer virus* in 1983. At that time, Dr. 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, suggested that he call his creation a *computer virus*. Dr. Cohen's Ph.D. thesis and later years of 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. 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 Apple II systems.

Although Cohen (and others, including Len Adleman) have attempted formal definitions of a 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 to copy files would be classed as "viruses" because it is possible to use them to copy themselves! 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.

Other forms of self-reproducing or 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:

> [*Back doors, Trapdoors*] Back doors, often called trapdoors, consist of code written into applications to grant special access by circumventing the normal methods of access authentication. They have been used for many years, and are generally written by application programmers who are seeking a method of debugging or monitoring code that they are developing. This usually occurs when a programmer is developing an application that

[2] Later reiterated in a letter to the editor of the New York Times, published in December of 1994.

has an authentication procedure, or a long setup requiring a user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges, or to avoid all the necessary setup and authentication. The programmer also may want to ensure that there is a method of activating the program should something go wrong with the authentication procedure that is being built into the application. The back door is code that either recognizes some special sequence of input, or is triggered by being run from a certain user ID. It then grants special access.

Back doors become threats when they are used by unscrupulous programmers to gain unauthorized access, or when the initial application developer forgets to remove the back door after the system has been debugged, and some other individual discovers its existence.

[*Logic Bombs*] Logic bombs are one of the oldest forms of malicious code. They usually are embedded in programs by software developers who have access to the code. A logic bomb is code that checks for a certain set of conditions to be present on the system. If those conditions are met, it executes some special function that is not an intended function of the code in which the logic bomb is embedded, and is not desired by the operator of the code.

Conditions that might trigger a logic bomb include the presence or absence of certain files, a particular day of the week, or a particular user running the application. It might examine to see which users are logged in, or which programs are currently in use on the system. Once triggered, a logic bomb may destroy or alter data, cause machine halts, or otherwise damage the system. In one classic example, a logic bomb checked for a certain employee ID number and then triggered if the ID failed to appear in two consecutive payroll calculations. A logic bomb embedded in a military system could be designed to disable to disrupt operations on a certain date, or if the code was being used in a particular country.

Of significant concern today is the significant use of commercial, off-the-shelf (COTS) software that has been produced, wholly or in part, outside the U.S. and/or using untrustworthy personnel. Many software vendors have notoriously poor source code control and testing procedures (viz., the large number of bugs and attacks against their products). Thus, logic bombs or hidden trapdoors included in their products are unlikely to be noticed or found. ***Software is regularly being used in mission-critical military and law enforcement tasks that has been produced under the control of individuals who would be prohibited from personally participating in those tasks. We do not adequately screen that software for unwanted, dangerous code. Many of us who work in information security see this as a major threat to U.S. national security.***

[*Worms*] Worms are another form of software that is often referred to by the term virus, especially by the uninformed. ¨So-called "cyberpunk" novels such as Neuromancer by

William Gibson refer to worms by the term "virus." The media has also often referred incorrectly to worms as viruses. The recent Slammer, CodeRed and ILoveYou incidents were all caused by software that is more correctly described as a worm.

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 necessarily change other programs, although they may carry other code that does, such as a true virus. It is this replication behavior that leads some people to believe that worms are a form of virus, especially those people using Cohen's formal definition of virus (which also would classify automated network patch programs as viruses).

In 1982, John Shoch and Jon Hupp of Xerox PARC (Palo Alto Research Center) described the first computer worms. 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. Drs. Shoch and Hupp named their own programs worms, because they saw a parallel to Brunner's tapeworms. The PARC 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 Morris Internet Worm of November 1988 is often cited as the canonical example of a damaging worm program. That worm clogged machines and networks as it spread out of control, replicating on thousands of machines around the Internet.

Few computer worms were written between 1988 and 1998, especially worms that have caused damage, because they were not easy to write by those inclined to want to write them for malicious purposes. Worms required a network environment and an author who was familiar not only with the network services and facilities, but also with the operating facilities required to support them once they reached a target. However, that dynamic began to change as vendors, particularly Microsoft, began to supply network applications with high-level macro interfaces. a could then use high-level macro constructs to write worms and viruses, and the network particulars were handled by the underlying applications (e.g., Outlook and Word).

[*Trojan Horses*] Trojan horses are named after the Trojan horse of myth and legend. Analogous to their namesake, they resemble a program that the user wishes to run — a game, a spreadsheet, or an editor. While the program appears to be doing what the user wants, it actually is doing something else entirely. For instance, the user may think that the program is a game. While it is printing messages about initializing databases and ask-

ing questions about "What do you want to name your player?" and "What level of difficulty do you want to play?" the program can actually be deleting files, reformatting a disk, or otherwise altering information. All the user sees, until it's too late, is the interface of a program that the user thinks he wants to run.

Trojan horses have been, unfortunately, common as jokes within some programming environments. They are often planted as cruel tricks on web sites and circulated among individuals as shared software. Note that the activity of a trojan is not necessarily damaging, but usually is unwanted.

[*Spyware*] Advertisers are continually seeking new ways to get their ads in front of potential buyers, and to collect information that could be used in marketing. One of the more annoying methods of doing this is to insert software into a user's operating system or browser that continually presents the user with pop-up ads. A quieter, but potentially more dangerous form of such software is *spyware* – software that records information about WWW sites visited and sometimes even as much as keystrokes typed. This information is then sent to a central monitoring site for analysis.

Most users are unaware that they have downloaded and installed spyware as part of the software they may be obtaining for other purposes. Usually, the purveyors of spyware include generic legal permission statements in the online license agreements that are presented to users when downloading software. Users seldom read these, or understand the full impact of what they mean.

Note that spyware used on sensitive systems may indeed be operated by actual spies, but disguised as commercial spyware!

[*Rootkits, exploit scripts*] When new faults ("bugs") are discovered in widely-deployed software, some individuals race to develop tools to exploit those flaws. These tools often contain sophisticated interfaces and documentation so as to enable unsophisticated users to employ them. These tools are then posted on newsgroups and WWW sites for open download. What results are widespread break-ins to sites where the patches for the affected flaws have not yet been applied.

The name *rootkit* derives from the goal of hacking into most Unix systems: obtaining access to the *root* account[3] . As some of these tools are written in simple scripting languages, the untrained people who employ them are known as *script kiddies*.

These kits and scripts are written by a variety of individuals. Some are well-meaning individuals who believe they are producing tools to help others determine vulnerabilities in

[3] The Unix root account is the superuser account. It is so named because the superuser owns the root of the file system rather than owning a particular named account.

their own systems. Some are simply antisocial individuals with ill-specified agendas, such as to cause embarrassment to particular software vendors. Often these exploits are an attempt to gain some form of notoriety in the marketplace.

[*DDOS, bots*] Systems that are designed to flood sites with more network traffic than they can handle are known as *denial of service* attacks, or DOS systems (not to be confused with MS-DOS or PC-DOS, the early PC programs). These first became a problem in late 1996. To heighten the effectiveness of these attacks, and to further obscure their origin, software has been constructed to create slave programs (robots, or *bots*) on compromised systems around the Internet. These bots maintain contact with a control channel, usually an Internet Relay Chat connection[4] . When the controller of the bots issues a command, all of the bots participate in a distributed denial of service attack, or *DDOS*.

There are a number of automated tools that scan large numbers of systems for vulnerabilities, compromise those systems, and then install bots for DDOS attacks. It is not unusual for thousands of machines to participate in DDOS attacks. The attacking hosts are difficult to trace, and the resulting network traffic can flood (or crash) multiple systems for hours or days at a time. One figure derived in 2001 using statistical methods suggests that thousands of these are occurring each week, although not all are severe enough to be noticed by victims.

Threats and Risks

The malware threat to U.S. systems, and the military in particular, is significant. Software is at the heart of most advanced weapon systems, command and control, communications, mission planning, and platform guidance. Intelligence, surveillance, and logistics all depend on massive computational resources. Less known but equally critical are the embedded processors and SCADA (system control and data acquisition) controllers that are used to adjust everything from flood control gates to utility distribution to building A/C controls. Disruption or compromise of any of these systems can significantly damage our national defense and public safety.

The U.S. military is highly trained and equipped. We have outstanding personnel and equipment. However, those personnel and their equipment are more dependent on correctly-functioning computational resources and communication than any military force in history. That we have equipped them with a computational infrastructure — in hardware and software — that is largely the same as anyone can buy from a major supermarket or mail-order house means that the core of their technical superiority is available for hands-on study by our opponents. ***Worse, a large hobbyist and civilian population is actively seeking weaknesses and attacks against exactly the same platforms used by our military, and they are sharing their findings on global mailing lists and WWW sites. Antagonists from lone fanatics to nation-states large and small have access to detailed information enabling them to construct effective weapons that target our IT systems.***

[4] Internet Relay Chat, or IRC, is form of distributed conferencing that allows users to exchange messages and files without any centralized control. It is similar to instant messaging.

The traditional model of security holds that three qualities need to be protected: confidentiality of information, integrity of data and software, and availability of service and data. The threats are counter to these, namely observation or disclosure of sensitive information, alteration or destruction of data or software, and denial or degradation of service. Traditional viruses target primarily integrity. DDOS tools target availability. Spyware accesses data and compromises confidentiality. Rootkits and backdoors provide access to privileged data and software on the system, thus compromising both confidentiality and integrity.

Currently, threats occur from a spectrum of antagonists. At some level, there are undoubtedly agents of foreign intelligence services and criminal organizations seeking information and mapping weaknesses. The level of this threat may not be accurately known because of the level of "noise" generated by the script kiddies and widespread DDOS attacks. Repeated experiments by groups such as the Honeynet Project have revealed widespread, automated scanning for target systems. Often a new, unpatched system will be compromised and a bot or backdoor installed within 15 minutes of it being placed online in the United States. I have heard of attack intervals as low as 90 seconds.

In addition to ongoing probes, marketing activities generate significant background noise. Unsolicited e-mail ("spam") accounts for as much as 70% of all network traffic in some environments. Some WWW-based probes are the consequence of visiting commercial sites. Some pop-up advertisements are permanently installed on systems through the installation of new run-time software, added without the user's permission. We are also seeing instances of advertisements that are actually worm programs. These worms install themselves on end-user machines and then proceed to send out spam e-mail using the new host, including copies of themselves.

The majority of these attacks are undoubtedly not directed against the U.S. as an entity, but the sheer volume of such traffic makes it difficult to distinguish actual hostile traffic from more benign activity. At the least, the volume of probes and spam is a significant degradation of service, thus meeting the definition of one form of "attack." It is well within the realm of possibility that this traffic is being used as camouflage by hostile actors.

There is a significant threat from simple failure that must not be overlooked. The complexity of our systems is increasing, and software (particularly commercial off-the-shelf or COTS products) are not developed to be robust in the face of active attacks and degraded environments. These factors may combine to cause unanticipated failures, with consequences beyond the ken of the operators. As more of this technology gets pushed into the hands of the individual warfighters, the likelihood of unanticipated and uncompensated failure will increase unless care is taken to simplify and harden the platforms. Use of COTS products optimized for running games and surfing the WWW is *not* likely to provide the necessary protection.

The insider threat is not being given enough consideration. At sites where strong network border

guards are in place and software is generally protected, a trusted insider can introduce dangerous malware that is designed to degrade or halt critical systems, silently corrupt data (e.g., change targeting and mapping information used in precision weapons), or disclose classified information. By being introduced on the inside, the software does not need to be written to overcome specialized protections, but only needs to establish itself on critical systems. This introduction can occur as a result of compromised software from a vendor or contractor, from a visitor or contract worker, from a disaffected or compromised employee or serviceman, or from coalition personnel with interests not in complete alignment with the US.

Enablers

Where malware has flourished is 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 defense databases and participating in networks of computer systems. Unfortunately, their hardware and older operating systems are still affected by 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 17 years after the first infection by the Brain virus in January 1986, the number of known viruses has grown to around 90,000 different viruses affecting Intel/Microsoft platforms. At any one time, approximately 500-1000 of those viruses are actually "in the wild" and posing a threat. The problem has not been restricted to the Intel/Windows PC, and now affects all popular personal computers. However, there are under 60 viruses that have ever been found for the Macintosh platform, and about a dozen for Unix-based platforms. This disparity reflects a number of factors, not least of which is the underlying software architecture of the operating systems in use.

Viruses may be written for any operating system that supports sharing of data and executable software, but all mainframe viruses reported to date have been experimental in nature, written by serious academic researchers in controlled environments. 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.

Eight years ago we saw the emergence of the macro virus. This is a virus written in a high-level macro language and attached to word-processing documents or spreadsheets. When an infected

document is opened on any computer platform supporting the software the macro is activated and spreads itself to other, similar documents on the system. As these documents are shared across networks, the macro viruses spread widely.

Originally discussed as a theoretical issue[5] , the first "in the wild" version appeared in late 1995. Microsoft distributed a CD-ROM to developers with the first virus for the Word program included by an unknown party. No public account has ever been given by Microsoft of how the virus came to be on the CD-ROM, or what they might have done to trace the author. The virus, since named the CONCEPT virus, quickly established itself and began to spread. Within 18 months, over 700 macro viruses had been circulated, and several vendors were indicating that macro viruses were the most commonly reported virus problem at customer sites. Macro and high-level viruses have become the most prevalent in the years since that time.

Unfortunately, macro viruses are here to stay. Users are loathe to do without their custom macros. Multimedia mail makes enclosure of infected documents simple and distribution even simpler. Increasing use of active content in WWW pages and automated downloads suggests that the problem will get worse as time goes on.

One of the biggest enablers of malware is the homogeneous nature of computing environments, especially in the military and government. Systems have been purchased with cost or compatibility as the defining criteria, and this has often included reuse of old software, hardware, and training. Thus, there has been a steady tendency to obtain systems from a limited set of vendor families. Because cost is an issue, COTS software is almost always at the base of these choices, despite the fact that COTS is not written for high reliability or security. Furthermore, the installed systems have their defenses set to lower than optimal to accommodate legacy software and peripherals that were designed for less-protected predecessor systems. The result is an infrastructure that has widespread vulnerabilities — a monoculture — and that is susceptible to widespread attack. If a vulnerability is discovered against one of these systems, there is an extremely high probability that it can be spread to many other systems in the same enterprise.

Consider this quote from a study[6] released by the Air Force Scientific Advisory Board in April 2000: "COTS software is not secure. … It is strongly recommended that COTS products, particularly software, not be used for critical applications."

The poor quality of most software is perhaps the biggest enabler of attacks against IT systems. Major, widespread attacks are enabled by the presence of significant flaws in deployed software. Those flaws are often the result of poor design and improper coding. Our studies have shown that over 70% of all published flaws in the last few years were caused by faulty coding practices that have been known for years, and often decades. Consider that the CERT/CC reported slightly under 2000 new vulnerabilities in the first half of 2003: that suggests that perhaps 1400 reported

[5] The late Dr. Harold Highland and I each made presentations on macro viruses at security conferences in 1991. Unfortunately, those conferences were never attended by representatives of the major software firms.
[6] Ensuring Successful Implementation of Commercial Items in Air Force Systems.

vulnerabilities (and all associated attacks) in that time were preventable by using known good methods of development.

Traditionally, code has been shipped without adequate testing or care taken in the design. Vendors have felt compelled to ship software with known flaws so as to compete "in Internet time" where time to market has been the most important criterion for success. Customers have largely accepted poor quality software rather than buy competing products that may cost more (and thus reflect the cost of producing higher quality code); the U.S. government is a prime example of this practice. The continual focus on lowest cost rather than ultimate fitness for use has discouraged companies from investing in better software engineering methods, and has also contributed to the increasing use of off-shore development and maintenance operations. Meanwhile, vendors have largely been immune from liability lawsuits despite negligent behavior. In fact, the software vendor community has sought to immunize itself from liability through mechanisms such as the UCITA[7] legislation put forward at the state level.

The result is that vendors of higher quality, safer software have found themselves serving a shrinking market – they face a significant penalty for spending extra resources to make their code reliable. Meanwhile, the typical system administrator may be faced with the prospect of installing and configuring as many as five critical security patches *per week* to the systems under her control. Each of these patches has the potential to disable 3rd-party software that is mission critical. However, the consequence of not installing a patch may well be a system break-in, or contamination of the system from a network worm, thus requiring a complete system scrub and rebuild. All of this is at the expense of the system operator. Unfortunately, this increased cost of operation is not included in the evaluation of price when the original purchase is made. Nor are the costs of of virus protections, firewalls, scanners, and other security tools that are not part of the base system but required to safely operate these complex systems.

To be fair, the vendors with a poor reputation for software quality simply have been reacting to the market. They are in business and must be competitive in the marketplace. As such, meeting customer pressure for low-cost, high-complexity code is what enables them to succeed. The fault for code quality problems lies with the consumers as well as the developers. Some companies have become quite sensitive to these problems and have initiated extensive programs to effect a change in quality control and security awareness. Microsoft's inititive in this respect is particularly notable.

Increased connectivity is also to blame for the magnitude of the current threat. Systems are configured so that every machine has network access. This is needed to provide for remote backups, access to patches, and user access to WWW browsing and e-mail. Unfortunately, that same access allows users without training to import and execute software and documents with macros. Once "inside" the security perimeter, malicious software can spread widely.

[7] UCITA is the Uniform Computer Information Transactions Act, an update of the Uniform Commercial Code that has been opposed by consumer advocates, professional associations, state attorney generals, the ABA and ALA, and many others. Its primary champions are large software firms.

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.

Defense against malware generally takes one of four forms, or as is more often the case, some combination of these four:

> [*Activity monitors*] Activity monitors are usually programs that are resident on the system. More general monitoring is now called *intrusion detection*, although it actually detects more than intrusions. These systems 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, send out many e-mail messages in a short amount of time, or to rewrite special portions of the disk would be intercepted by such monitors. This form of defense can be circumvented by malware that activates earlier in the boot sequence than the monitor code. Many rootkits and viruses contain code that is designed to alter the operating system so as to hide from activity monitors.
>
> [*Scanners*] Scanners have been the most popular and widespread form of malware 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. Other forms of scanners look for known signs of rootkits or intrusions, and also may look for known vulnerabilities that might be exploited by such software. It is usually the case that virus scanners are separate programs from the more general form of security scanners.
>
> 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. New viruses are appearing by as many as several dozen each day. Keeping a pattern file up-to-date in this rapidly changing environment is difficult. Although it is unlikely that any given user will encounter any particular virus, a single activation by a machine in a critical environment can be devastating.
>
> 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 some self-altering viruses cannot easily be detected with scanners.
>
> To the advantage of scanners, however, is their speed. Scanning can be made to work rea-

sonably quickly. Scanning can also be done portably and across platforms, and pattern files are easy to distribute and update. Furthermore, of the new viruses reported each week, few will ever become widespread. Thus, somewhat out-of-date pattern files are still adequate for most environments. It is for these reasons that scanners are the most widely-used form of antivirus software.

A variation on scanners that is used by some vendors is heuristic scanning. In this case, new code is examined instruction by instruction to determine if it matches any known pattern of behavior that is common to viruses or other malicious software. This technique can be effective against previously unseen virus code, but it also tends to have a high false positive rate, thus requiring manual intervention.

[*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. 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.

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 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.

[*Border guards, firewalls, proxies*] These are software/hardware combinations that are placed at gateways and borders of networks to examine all traffic into a network. These

systems look for known attacks, viruses, and other dangerous content. Some also scan for prohibited items such as pornographic pictures. When content is found, it is interdicted.

Border scanners are a help in many environments, but they fail when scanning encrypted contents, such as in encrypted e-mail and VPNs (virtual private networks, or tunnels). They also fail against previously unseen content, or when users actively seek to circumvent them. This often happens when a user is seeking to obtain prohibited material, and unknowingly brings in a trojan horse artifact.

There are some experimental systems that seek to measure untoward network behavior and isolate machines that are behaving in an anomalous manner. Automated measures at a larger scale may be necessary to cope with the increasing virulence and speed of malware. Consider:

- The Brain virus, introduced in 1986, required 5 years to reach its maximum level of spread. This was to approximately 50,000 machines, and resulted in perhaps $5 million in damages according to some estimates.
- The Melissa macro worm, released 13 years later, spread to approximately 150,000 systems over a period of four days. Damage was estimated to be in the vicinity of $300 million.
- The ILOVEYOU macro worm, released in May 2000 spread to as many as 500,000 systems in a little over 24 hours. Damage was estimated to be as much as $10 billion.
- The Code Red and Nimda worms in October/November 2001 exploited flaws with published fixes but still managed to compromise 500,000 systems in 14–16 hours. Several billion dollars in damages were estimated.
- The Sapphire/Slammer worm at the beginning of this year, also exploiting flaws with known patches, reached its maximum spread of 75,000 systems in 10 minutes. It was doubling every 8 seconds. It caused over a billion dollars in damages (approximately $13,000 per machine; $1.7 million per second).

Faster propagation of malicious software is possible, especially if some preplanning is done, and it is started by multiple entities. Greater damage is also possible.

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.

In addition to the virus problem is the ongoing problem with DDOS, rootkits, trojan horses, and other attacks. The CERT/CC recorded over 82,000 major attack reports for 2002. In the first half of 2003 they have reported over 76,000. Analysts at Symantec Corporation have esti-

mated that worldwide there were over 80,000 network intrusion attempts in 2002, and over 800 million attempted virus infections. At the current rate of growth, these are expected to reach 100,000 and 120 million, respectively, this year. These are not salutary trends.

Defense against Trojan horse programs, rootkits, and logic bombs is generally limited to intrusion detection systems, firewalls and code inspection. Intrusion detection systems examine log files and/or network traffic to detect known patterns belonging to known attacks or suspicious activity. New attacks, or gradual attacks are often not detected. Firewalls then provide the next level of protection by denying access to certain network services and ports based on policy and need. Unfortunately, users often circumvent these protections with "tunnels" or "proxies" because the firewalls prevent access to desired services. Additionally, tuning of the firewall policies is not simple, and small mistakes or oversights often lead to problems. And finally, if there are flaws in services that are supposed to be exposed to the outside network, the firewalls provide no protection.

Code scanning is a class of techniques used to ensure that software imported to a machine is free of malicious code. This may constitute scanning with automated tools to look for known flaws, or it may involve a more formal procedure of examination such as is done with the Common Criteria. Unfortunately, these examinations are often limited in scope, require some cooperation of the software vendor, require significant time and expense to complete, and are not designed to search for all possible flaws. As the examinations are not carried out after each upgrade and patch, it is still possible to insert malicious code into otherwise protected systems. With some commercial operating systems with applications and database systems installed comprising close to 100 million lines of source code, any examination process using current technology is bound to be incomplete.

Unfortunately, there appears to be no lessening of computer virus and hacking activity. Many new viruses are appearing every day. Major flaws and corresponding attacks are reported every few days. 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.

Legal Issues

It is very difficult to track computer viruses once they have established themselves. Some luck may be had with tracking a computer virus to its authors if it is found very early after its release. To date, there have been only about seven publicized cases of authors being arrested, tried, and convicted for releasing viruses or similar malware. In most cases, the convictions carried only a fine and a suspended sentence. *For this to be the only visible punishment for over 20 years of virus-writing and almost 100,000 viruses written speaks to the difficulty of coping with the problem within established legal structures.* The little experience we have had with these cases also suggests that the convictions did little to dissuade others from writing viruses.

The same problem occurs with the variety of software break-ins that occur. Each case currently requires investigators with training beyond the norm, access to specialized forensic labs, and (often) cooperation of agencies in foreign jurisdictions. Investigation and then prosecution of computer crimes is vastly underfunded and understaffed in the U.S. today. Each case is expensive to pursue, and often the damages do not justify it. When juveniles are involved, or transnational jurisdictions, there is even less incentive to pursue such cases. The result is a lack of deterrence, and this leads to a continuing high level of attack against critical systems. These attacks draw away resources, and help mask more sinister activities that may be occurring.

The writing of computer malware is not a crime in most places. It is arguable whether writing a virus or attack tool should be a crime, exactly as constructing a bow and arrow is not innately a crime in most jurisdictions. It is the use of the item, and the state of mind of the user that determine the criminality. As such, it is probably the case that the deliberate release of a computer virus should be considered criminal and not simply the writing of the virus. Laws should reflect that difference. However, lawmakers have discovered the same difficulty in clearly defining a virus that researchers have encountered. An overbroad definition such as Cohen's would make the authoring and release of almost any software illegal; the presence of bad laws hurt the situation more than help it, especially when some of the same techniques are used in writing protective software and building test platforms.

The difficulties posed by laws against writing any kind of software is best illustrated with what has happened with regards to copyright. As more content has been developed for use with computers and networks, there has been a greater concern for protecting intellectual property represented by that content. Content owners have stridently lobbied for greater and greater protections for their on-line property. Unfortunately, the evolution of the law has led to unintended consequences for those of us working in security. In particular, I have heard of several instances where research into novel forms of information security have been curtailed because patent holders have threatened researchers. University faculty members do not have the resources to fight such threats.

More recently, provisions of the Digital Millennium Copyright Act (DMCA) have led to faculty being threatened with lawsuits for publishing their security research, and some faculty (Fred Cohen and myself included) have decided to curtail or stop our research in some areas of security because of the potential for us to be arrested or sued. This is particularly true in the area of software threats — the very same tools and techniques necessary to reverse-engineer and protect against malicious software are seen as a threat by many in the entertainment and content provision industries. ***Legislation against technology instead of against infringing behavior can only hurt our progress in securing the infrastructure.***

Some Recommendations

There are several actions that can be taken to reduce the threat of computer malware in the government and military. All of these can be derived by examining the problems that confront us. Among those that have the highest likelihood of making a difference, I would include:

1. Explicitly seek to creating heterogeneous environments so that common avenues of attack are not present. This *may* require some extra expense *at first,* but eventually it may lead to increased compliance with standards, increased innovation, and increased choice in the marketplace, thus lowering costs while increasing security. If real standards (rather than de facto standards) are developed and followed, interoperability should not be a concern.
2. Complementary to the previous recommendation is giving thought to different architectures. Rather than a computer on each desktop, thin-client technologies based on a mid-size computer in a centralized location can provide all the same mission-critical services, but remove many of the dangerous aspects of distributed PCs. For instance, patches need only be applied in one location, and there is a greatly reduced possibility of untrained users loading untested media or software.
3. Rethink the use of COTS software in mission-critical circumstances — the lowest cost is not necessarily the most fit for use. At the least, investigate better methods of screening and testing such software to ensure that it does not contain hidden, unwanted code. At the same time, hold the vendors to a higher standard of care and responsibility for what is in their code.
4. Rethink the need to have all systems connected to the network. Standalone systems may not receive all of the latest patches as soon as they come out. However, that alacrity may not be needed as those systems can no longer be attacked over the network.
5. Require greater efforts to educate personnel on the dangers of using unauthorized code, or of changing the settings on the computers they use. It is still often the case that personnel will turn off security features because they feel it slows them down or gets in their way. Unfortunately, this can lead to significant vulnerabilities.
6. Revisit laws, such as the DMCA, that criminalize technology instead of behavior. It is extremely counterproductive in the long run to prohibit the technologists and educators from building tools and studying threats when the "bad guys" will not feel compelled to respect such prohibitions.
7. Provide increased support to law enforcement for tools to track malware, and to support the investigation and prosecution of those who write malicious software and attack systems. This includes support for additional R&D for forensic tools and technologies.
8. Do not be fooled by the "open source is more secure" advocates. Whether source is open or proprietary is not what makes software reliable. Rather, it is the care used to design and build it, the tools used to construct and test it, and the education of the people deploying it. In fact, some Linux distributions have had more security flaws announced for them in the last 18 months than several proprietary systems. However, some open source software, such as OpenBSD and Apache, appear to be far more reliable than most proprietary counterparts. There is no silver bullet for problems of quality and security.

9. Initiate research into the development of metrics for security and risk. Acquiring systems based on cost as the primary criterion is not reasonable for mission-critical applications. We need to be able to differentiate among different vendor solutions, and set standards of performance.
10. Establish research into methods of better, more affordable software engineering, and how to build reliable systems from untrusted components. 15-20 years ago the decision was made to cede research in this arena to the commercial sector, believing the market would drive innovation. That has not happened. The military needs to reengage in this domain to ensure that their unique and their critical needs are met.
11. Emphasize the need for a systems-level view of information security. Assuring individual components does little to assure overall implementation and use. This requires trained personnel with an understanding of the "big picture" of IT security. Too often those who design and specify the systems do not understand how they are actually used....or misused.
12. Establish better incentives for security. The current climate in many military commands and government agencies is to penalize operators for flaws, thus leading many of them to dread enhancement and exploration of better security.
13. Increase the priority and funding for basic scientific research into issues of security and protection of software. Too much money is being spent on upgrading patches and not enough is being spent on fundamental research by qualified personnel. There are too few researchers in the country who understand the issues of information security, and too many of them are unable to find funding to support fundamental research. This is the case at our military research labs, commercial labs, and at our university research centers.
14. ***Most importantly***, reexamine the issues of the insider threat to mission critical systems – from obtaining software produced by uncleared personnel offshore and in this country, from using COTS products that are not designed for security and reliability, and from access and operation by untrained or unsupervised personnel.

Conclusion

1. It is clear that we have deficiencies in our cyber defenses. Malicious and incorrect software pose particular threats because of their asymmetric potential — small operators can exercise large and devastating attacks on our defenses. The situation cannot be remedied simply by continuing to spend more on newer models of the same systems and defenses that are currently deficient. It will require vision and willingness to make hard choices to equip our military with the defensible IT systems they deserve.

I will be happy to expand on any of these points, now or in the future.

Thank you again for the opportunity to testify.

Acknowledgments
I received many suggestions from colleagues when composing this testimony. I wish to acknowledge the people listed for their assistance. However, the content and opinions expressed are my own, and the presence of these names should not be construed as endorsement of any of the statements herein.
Rebecca Bace, John Reel, Paul Barry, Terry Kelly, John Davis, Robin Roberts, Kenneth Olthoff, David Isacoff, Paul Williams, Sarah Gordon, Annie Antón, Dwayne Melancon, and Mark Bruhn. I also received statistical information from Symantec Corporation, and from Tripwire Corporation.