Essay 4

Malicious Software

Marshall D. Abrams and Harold J. Podell

Risk — Possibility of system failure

Malicious software is one of the concerns of the contemporary computing environment. Malcontents, pranksters, vandals, and adversaries all have the potential ability to disrupt the conduct of our computing business. Prudence dictates that we analyze the situation and take appropriate countermeasures.

As with other concerns, the first responsive step is to perform a risk analysis. In the risk analysis we determine the threats and vulnerabilities, risk, effects, and countermeasures. That is, we conduct a study and make a managerial decision about what harm could occur to our computing environment, how likely it is that this harm will occur, what we could do to reduce the probability of this harm occurring, and what the impact would be if some harm did occur.

We must know what management's information technology (IT) and IT security policies are for the organization to establish the framework for assessing the results of a risk analysis. If we know the provisions of the applicable IT security policy, then we are in a better position to assess the relative importance of the countermeasures available to protect against the perceived threats. Money also plays an important part in our thinking. But there are also administrative and physical controls that could be implemented which have no direct cost, although they may have undesirable side effects in terms of convenience and productivity.

There are many terms for malicious software: Trojan horse, virus, worm, trap door, time/logic bomb, and so on. Some authors have spent considerable effort in developing precise taxonomies [EICH89, SPAF89, SPAF90a-c]. In this essay we use "virus" as a general term. If important, we will identify the particular malevolency by name or characteristic. Although interesting, we do not see much security value in overly precise semantics. Our focus is on identifying threats, vulnerabilities, and risk, and on providing countermeasures. We do not discuss in detail all

types and varieties of malicious software or subtle differences and similarities between terms. Since the appearance of new malicious software is unpredictably dynamic, we refer the interested reader to bulletin boards or other publications of current threats.

Popular attention to viruses highlights a real problem. Computers are under attack. Malicious software has been released to work its harm. The damage worked by a virus can range from a gross denial-of-service attack to a subtle modification of stored data and programs that will result in future malfunction. Sometimes the attack is selectively targeted; more often it is undirected. Sometimes the attack results from a benign error or a misguided attempt to call attention to some cause or fact of importance to the originator. More often, the originator has an antisocial objective. Although at one time there may have been some ambivalence toward the creation of viruses, by now it is (or should be) well understood that such activity is in violation of most codes of computing ethics [FORC90].

The scope of an attack ranges from a single computer to international destinations, such as many computers connected to the well-known Internet. We are most concerned about the potential damage from future attacks that are more sophisticated than any seen to date. What would happen if virus attacks were a weapon of national or industrial warfare?

The possible impacts of a virus attack are limited by the imagination and competence (evil genius) of the perpetrator. Impacts can include destruction or modification of data, unavailability of computing resources, interruption of operations, fraud and other financial crime, embarrassment, or loss of life.

The writers of malicious programs are generally anonymous, but occasionally a virus writer is identified. Often the authors' desire for publicity leads them to self-identification. For example, the Macintosh virus "Universal Message of Peace" displays the name of Richard Brandow, the Montreal editor of *Macmag* who commissioned the virus [STEF90]. Drew Davidson, who wrote the virus, left his name in the code. In addition to its human interest value, knowledge about virus writers can help us improve our preventive measures.

We should clearly understand the limitations of reasoning by analogy. Infectious diseases spontaneously mutate. New strains that are resistant to previously effective countermeasures can be dangerously effective. Countering the attacks of malicious software much more closely follows the model of war against crime or a political enemy. It is the opposition of human intelligence and skill.

Case studies are instructive in dealing with an attack situation. They teach us how damage was done in the past and what countermeasures proved effective — and may even point to future attacks. When attacks follow a known pattern, we can mount effective defenses. We can use

the power of the computer to (attempt to) search for, identify, block, and disable known attacks. We can apply similar techniques against generic types of attacks. But a truly innovative attack will probably surprise the defenders and be successful. Therefore, part of our defensive strategy must include damage assessment and recovery plans. In fact, we may decide to rely more on recovery than on prevention.

We study prior attacks as part of an engineering approach to learn from the past. It is easier to relate to an actual event than to abstractions. Case studies prove that the problem is real and provide psychological insight that may be helpful in constructing future defenses.

Publicity, especially in the press, focuses community attention on the problem. Managers need to understand that malicious software is a social phenomenon. It is not embarrassing to have been attacked, but it is a dereliction of duty to have been unprepared to deal with the situation. Taking no action in advance is itself evidence of having made a decision, albeit a questionable decision. Sticking one's head in the sand is not generally considered prudent management. The managerial response to malicious software attacks requires competent technical advice and support.

Definitions

The Trojan horse is a very common type of malicious software; it was probably the first. The *-property in the Bell-LaPadula policy model was designed to thwart the Trojan horse attack of illicit downgrading of classified information. The Trojan horse's modus operandi is that it appears to be an ordinary useful program which the user runs. Once in execution, the program runs with whatever privileges the user possesses. Most Trojan horses perform the desired function for the user. They often masquerade as legitimate programs or are malicious variants of legitimate programs. They often entice the user to execute them by promising some improvement over the legitimate program they replace. One-time-use Trojan horses only perform malicious activity; more sophisticated multiuse Trojan horses perform their advertised useful function while also surreptitiously performing malicious activity.

Protective countermeasures such as identification and authentication may have been used to keep out unauthorized people, but most systems have no means of identifying and authorizing trustworthy programs. There are research efforts that extend authorization to programs. Biba [BIBA77], Schell and Shirley [SHIR81], and Clark and Wilson [CLAR87] all have made contributions to this form of control. Strict configuration management practices are also useful in limiting the introduction of Trojan horses. The weak link is usually the human user, who is often unable to resist the temptation to run a faster, better, pirated, free, or test version of some game or useful software.

As mentioned, we do not offer overly precise semantics for definitions of malicious software types. For purposes of this discussion, we use generic definitions of a virus and a worm. A computer virus is usually constructed with two objectives. The first is to replicate. For example, a virus may copy itself into a useful program. A virus may invade system files and replicate itself. Second, a virus has a specific function (or functions) defined by the virus writer. This second objective could include displaying a message, erasing sectors on a hard disk, or expanding until it slows down other processes in the computer. A worm that is malicious may be considered a network extension of a virus that uses a network communication mechanism for propagation.

A virus can infect other programs by modifying them to include a copy of itself. The infected program in turn infects other programs. The infection spreads at a geometric rate. One symptom of a virus infection is a change in the length of a program. Some poorly designed viruses continue to install themselves in the same program, causing unlimited growth. This defect makes identification of such viruses a particularly easy job.

Time/logic bombs are malicious code that is executed when a certain event occurs, such as Friday the 13th. Trojan horses and viruses may contain time/logic bombs.

There is really no such thing as a harmless virus. Even if a virus was intended to cause no damage, it may do so in certain cases, often due to the incompetence of the virus writer. A virus may be modified, either by the original author or by someone else.

Threats — Possible attacks

A well-studied threat to information security is disclosure. The protection of secret information has always been part of diplomacy and warfare. Protection against unauthorized disclosure of information stored in computer systems was a direct generalization of manual methods used to protect similar information written on paper. In essence, when the computer replaced written records and communication, the protection policy was adapted to this new medium. Unauthorized disclosure can result from malicious software, such as a Trojan horse. Classical protection policy and mechanisms already exist to counter this threat.

In contrast, malicious software represents an attack on the computer itself. Existing policy is inadequate; the nature of the attack is understood but imperfectly. The theoretical models for protective mechanisms are still evolving.

An important threat to data integrityis unauthorized modification of information stored in computer memory. This information includes data, especially programs. The operating system and productivity programs are typical targets because of their ubiquitous potential to spread the harm.

Threats to resources include unauthorized utilization, destruction of information, and denial of service. Unauthorized utilization often results in theft of service. Time-sharing, information service, and electronic mail/bulletin board vendors all expect to be paid for the use of their services. Unauthorized unpaid use of such services is widely recognized as a crime. The damage caused by a virus may consist of the deletion of data or programs, maybe even reformatting of the hard disk, but more subtle damage is also possible. Some viruses may modify data; the amount and type of damage are limited only by the imagination, competence, and access of the perpetrator.

Destruction of information is an extreme case of unauthorized modification. It is probably easier for the malefactor to delete information than to change it, but the effect of subtly altered information can be much more devastating. The integrity of the system can be compromised, including databases and programs — especially system software and applications. If information is missing, it can be reloaded from backup media. But if it is maliciously altered, it could be used to produce erroneous results in far-ranging applications. For example, a change in the value of pi could cause operational errors. In severe cases, these errors could result in the crash of a satellite.

Denial of service can be a primary objective or an unintended by-product. For example, if a computer system is only performing operating system functions, there could be a denial of service to legitimate system users. This resource consumption can slow or stop information system operations. Communications networks are especially susceptible to such attacks. A flood of housekeeping messages, even maintenance messages or malfunction alarms, can make the network unavailable to perform its intended function. Such a flood of messages can be produced deliberately or can result from a design error in benign or malevolent software.

Virus vulnerabilities — Managerial responsibilities

Computer systems are often vulnerable to attack by malicious software because well-known protective measures are not implemented. For a privately owned personal computer, the responsibility is clearly on the owner. Since it is this owner who suffers the effect of an attack, one occurrence should be more than sufficient to induce self-protective behavior.

For organizational computing, there is a managerial responsibility to encourage, if not enforce, safe computing. The individual who has custody of an information asset may not be in a position to judge the value of that asset and may, therefore, not adequately protect the asset. Or the individual may make his or her own trade-off analysis of the risk, value of the asset, and effort required for protective measures. The indi-

vidual's judgment may differ from management's. It is management's responsibility to specify the tools and procedures to be used to protect the organization's information assets and to provide the time and resources to enable employees to carry out the procedures.

Lack of management attention usually results in a lack of protection. Protective measures take time and effort to install and use. There are direct costs for purchase, storage, and labor. In contrast with centralized computer operations where security responsibility could be assigned to individuals and made an explicit part of their job subject to supervision and performance appraisal, distributed computing environments often place all responsibility on the individual who is the direct user of the computing equipment. These end users, be they members of the secretarial or professional staff, should be assigned specific responsibilities for protecting the information assets under their control.

In addition, open distributed processing (ODP) implies the existence of several sets of end users for business systems that support electronic commerce. For example, the enterprise viewpoint considers user interfaces that may be interpreted to include business requirements. ODP business requirements include electronic interfaces with end users outside the organization, such as customers and suppliers that have computer systems for EDI (Electronic Data Interchange).

When network communications are involved, management responsibility includes security operations. There is extensive connectivity in networks. Most networks also lack security control. Most came into existence with concern for functionality, not security. Small networks, especially local area networks (LANs), using homogeneous equipment and protocols and operating under the control of a single management scheme, can institute many security measures. In many ways LANs resemble individual computer systems. User programming and policy inadequacy are among the problems to be considered.

Large networks, or internets, created by connecting LANs with wide area networks (WANs), usually cannot be relied upon to provide adequate security services. The security must be provided by the LANs or the end systems — the computers used to provide user service. Management is responsible for determining the threat introduced by the network connection, the secure protocols and other measures that are available, and the extent to which these protocols and measures are used by other management domains accessible through the network. Given this information, management can then decide whether to connect to the WAN and which protective measures and protocols to use.

Malicious software threats

Stand-alone computers include single-user desktop computers and workstations. The minimum set of malicious software problems occurs

in stand-alone computers: The use of corrupted programs and lax security are typical of open, trusting environments. Many users are unsophisticated about computers in general and unwilling to take precautions they consider too expensive, too technical, or too time-consuming or otherwise obtrusive into their work patterns.

Multiuser systems are also subject to irresponsible (or worse) activity by authorized users. In fact, protection of one user against malevolent or accidental damage by other users was the original thrust of computer security.

Computers connected directly or indirectly to WANs are susceptible to remote attack. Unfortunately, this evil potential makes an open sharing policy naive. Viruses work best when sharing in a network is inadequately controlled.

Network complexity offers considerable opportunity for errors, omissions, and exploitation beyond any one person's, or one organization's, control. There is continuing increasing investment in international networks. Major functions are being delegated to distributed processing systems in international commerce. Essays 17 and 18 discuss Privacy Enhanced Mail (PEM) and Electronic Data Interchange (EDI). These sophisticated applications require end-system/network security for user authentication, message authentication, and integrity. Encryption mechanisms are widely used as security countermeasures. (See Essays 15 and 17 on this point.)

Trojan horses need covert channels. Trojan horses can attempt to violate multilevel security by communicating classified information to a process below the classification level at which the Trojan horse is running. Remember that a Trojan horse operates with all the privileges of the process representing a user, so it is easy for the Trojan horse to obtain copies of information that the user processes. Multilevel security policies generally prevent a user from writing information at a lower level than the one at which it is read. In the Bell-LaPadula policy, this restriction is enforced by the *-property. In fact, the *-property was introduced more to block Trojan horses than to prevent user errors. When the Trojan horse is blocked by the *-property from illicitly downgrading classified information, it must use more sophisticated indirect methods to communicate information down to a less-classified accomplice process. One such method is the covert channel.

A covert channel is a means of signaling information from one user to another using a mechanism that was not designed as a communications channel. If one process can change the state of some system characteristic that another process can sense (in violation of the security policy), then a covert channel exists between the two processes. The system characteristic could be a shared variable, such as a register, or it could be the rate at which the receiver process was able to transfer informa-

tion between primary and secondary storage. The former is an example of what is conventionally called a covert storage channel and the latter of a covert timing channel. Recent thinking is that all covert channels have both elements of timing and storage, reducing — if not eliminating — the distinction between types of covert channels.

Trap doors. Trap doors in software permit entry without detection. They are used by system designers for ease of entry; that is, the trap doors are inserted in the system by people in a position of trust to permit these people to bypass the system's protective mechanisms. Trap doors are often rationalized as necessary to permit access when the system has gone into an undesirable error state and all other means of access are blocked. Perhaps trap doors are necessary as a development tool, but when they are left in operational systems, it is relatively easy for penetrators to find and use them for unsanctioned entry.

Personal computer viruses. Two different types of viruses occur in PCs: boot sector viruses (BSV) and program viruses. A BSV infects the boot sector on a diskette. Normally the boot sector contains code to load the operating system files. The BSV replaces the original boot sector with itself and stores the original boot sector somewhere else on the diskette. When a computer is then later booted from this diskette, the virus takes control and hides in RAM. It will then load and execute the original boot sector, and from then on everything will be as usual. Except, of course, that every diskette inserted in the computer will be infected with the virus, unless it is write-protected. A BSV will usually hide at the top of memory, reducing the amount of memory that DOS sees. For example, a computer with 640K might appear to have only 639K. Some BSVs are also able to infect hard disks, where the process is similar to that described above.

Program viruses, the second type of computer viruses, infect executable programs. An infected program will contain a copy of the virus, usually at the end, but in some cases at the beginning of the original program. When an infected program is run, the virus may stay resident in memory and infect every program run. Viruses using this method to spread the infection are called "resident viruses." Other viruses may search for a new file to infect when an infected program is executed. The virus then transfers control to the original program. Viruses using this method to spread the infection are called "direct action viruses." It is possible for a virus to use both methods of infection.

In general, viruses are rather unusual programs — rather simple, but written just like any other program. It does not take a genius to write one — any average assembly language programmer can easily do it. Fortunately, few of them do.

Observations about network viruses

Network e-mail connectivity is important for reporting problems and getting bug fixes; disconnecting e-mail connectivity can cripple information flow. Networks that are loose confederations without central management find it difficult to respond to attacks. Even corporate networks may not have planned for attacks. When there are no plans for resisting an attack, personal contacts work well. Of course, there are problems at 3 a.m. in locating a responsible person at a remote site and authenticating yourself, especially if the normal mode of communication is e-mail and you are using the telephone.

Analyzing the attack and determining how to thwart it are difficult, intense intellectual activities. Misinformation and illusions run rampant. Source code availability is essential to understanding how the attack works. Vendors who do not release source code thereby take on full responsibility for fighting attacks. Marshaling the people with necessary skills is most easily done in an academic environment, which has such people available to be diverted from their regular tasks.

Lessons learned from fighting attacks are accumulated for future use [ROCH89]. Least privilege is one of the security approaches to reduce a computer system's exposure to attack. Users should be given all the privileges needed to do their work, but no more! Ignoring this fundamental principle frequently leads to disaster. Diversity is good; attacks are usually specific to an operating system and implementation of standard protocols. That is, malicious software exploits weakness in targeted design and code. The cure shouldn't be worse than the disease. The cost and inconvenience of countermeasures should be less than the effect of a successful attack. Denial-of-service attacks, in particular, are easy to launch but difficult to prevent. In the limit, it may be more expensive to prevent certain types of attack than to recover from them. In any case, local and remote backups, audit records, and archives are essential to recovery. Since preventive measures cannot be assumed to be 100 percent effective, the ability to recover system state prior to the attack is prudent. This recovery can often be facilitated by the use of redundant network and network security architectures.

Defenses must be at the host or end system level, not the network level. In terms of the OSI protocol layers, it is possible to implement some countermeasures in layer 3, but most are situated in higher layers. Any countermeasure that relies on least privilege and individual user identification must be implemented in the end system at layer 6 or 7. In other words, computers connected to networks must defend themselves from threats carried by the networks. It is not possible for the network to provide protection against each class of attack.

Logging of audit information is important. The more information that is specifically available about an attack, the easier it is to diagnose and

neutralize. The audit trail can also make it possible to identify the source of an attack. However, too much information in an audit trail can make the resulting analysis very challenging. Electronic analysis of trends in the audit records can assist in the analysis.

A central repository of identified attacks and countermeasures is a good idea. But there are practical problems. Vendors must participate. When source code is proprietary, the vendors may be the only ones able to implement countermeasures. When a business relationship exits between the vendor and the customer, the vendor can provide security-related information to these customers, such as the existence of flaws and the appropriate fixes. But not all customers maintain a relationship with the vendors of their hardware and software. Equipment is resold or otherwise made untraceable. Even if the vendor's behavior is exemplary, the users may not be sufficiently security conscious. Fixes are often not installed because they are not identified as security related, or because the users have not experienced any problem, or because the installation is too difficult. Some users may even be malevolent; distributing information about security flaws could inspire further attacks.

Countermeasures

Our discussion of countermeasures is organized in order of detection, reduction, recommendations and observations, and legal remedies.

Procedural and administrative controls. One place to attack malicious code is at the source. Organizations need to guard against malfeasance on the part of their employees. Modern software development methods are quite useful for reducing the incidence of malicious code being inserted during the production cycle [AMOR91]. Division of labor specifies that programmers do not write their own program specifications. Code reuse assures that well-tested and debugged software is used in preference to writing new code. Of course, if reused code contains a virus, reuse assures its further dissemination. When software tools are prescribed, the programming environment is specified; this also reduces the freedom to introduce illicit code into production software. The formal team approach provides for peer review of software modules, independent testing, and configuration management.

Operational software, as well as software in development, is protected by the integritycontrols of configuration management (CM). Configuration management includes a formal set of procedures to control all program modifications. The objectives of CM are to maintain program integrity, evaluate value and correctness of all proposed modifications, and exercise administrative control on test and installation procedures. It should be very difficult to insert malicious code into a system under CM.

Other administrative and physical integrity controls can be used to reduce vulnerability to attack. Management must decide which controls are applicable in their environment. One obvious step is to put all controlled software on read-only media. Note, however, that assurance of media write protection may not be as simple as it appears. Many forms of write protection use a hardware sensor to detect the presence of a physical marker, but the enforcement is left to software. Malicious code might be able to bypass this software protection. A physical switch that prevents write current from flowing is much more reliable.

Software engineering standards include life-cycle management functions. CM is one common control structure for life-cycle management, but it pertains almost exclusively to control of production code and documentation. The software life cycle also includes requirements definition, design, development, and testing. Security really needs to be considered in all of these phases. For example, policy decisions such as whether to permit program development, the use of software downloaded from bulletin boards and archives, or anonymous login and file transfer all have significant impact on the risk environment. These policy decisions are part of the administrative security measures.

Another name given to a class of countermeasures is internal controls. This term comes from the audit community, where it pertains to providing for financial and program integrityThere is an overlap of terminology; it is only necessary to note that internal controls are also countermeasures to malicious programs.

The principle of least privilege is simple in concept: Give people the privileges they need to do their jobs, but no more. In practice it may be difficult to determine what privileges are needed, or to dynamically change these privileges in coordination with changes in job assignments. Most observers agree, however, that the extreme of giving certain users unrestricted privileges is dangerously naive. The existence of an all-powerful "superuser" is dangerous unless benevolence and omnipotence can be guaranteed. Since this is unlikely, safe computing practice is to eliminate the "superuser" by such administrative techniques as separation of duties and rotating assignments.

PC virus protection programs. There are many programs available that offer some degree of protection against malicious code. In selecting a protective program, you should consider the type of detection. That is, upon what criteria does the antiviral program decide that a virus has been found? Antiviral programs are generally divided into three categories: filters, change checkers, and scanners.

A filter is a program that monitors the system for virus-like activity (that is, attempting to format a hard disk, write to a program file, and so forth). Filters have the advantage of being able to detect new viruses because they are not looking for specific viruses, but rather for suspi-

cious activity. The disadvantage is that they can be prone to false alarms triggered by programs that perform potentially dangerous activities for legitimate reasons.

A change checker computes a checkfunction reference value for a protected file and is subsequently run to compare the current value against the reference. If the reference value and the just-computed value don't match, then the file has been modified and may be infected with a virus or otherwise subjected to unauthorized modification. Like the filters, change checkers will detect known and unknown malicious activity. They are not checking for specific pieces of code, but rather for changes to a computed value. This method works only if the change checker is installed in a virus-free machine. Otherwise, the reference values computed will reflect the installed malicious code.

A scanner works by checking the system for pieces of code unique to each virus. The scanner reads the files (boot sector, partition table, and so on) of a disk and does a match against a database of bytes that are segments of code unique to each virus. When a match occurs, a virus is reported. This is effective for finding known viruses, since a positive identification of the virus is made. Of course, a false alarm could also occur if a file had the same instructions in it. Scanners can also check for generic routines, like a series of program instructions to format a disk, but these could raise a false alarm concerning a program or routine for legitimate purposes. A scanner will detect only known viruses and must be updated frequently; as more viruses are added, the scanner gets slower. Scanners may also be ineffective against viruses hidden in compressed files, especially if multiple compression is used.

Technical integrity controls. Many malicious software attacks result in unauthorized modification, which is an integrity violation. Checksums provide an indication of unauthorized modification. A checksum is calculated by treating the source — program or data — as numerical. The numerical values are summed (over some modulus), the result being the checksum. When a function more complicated than simple addition is used, the result is known as a checkfunction or a hash sum. It is very difficult to modify software and preserve the value of the checksum. Cryptographic checkfunctions are among the strongest. A type of cryptographic checkfunction is called a digest of a message or a message digest. As discussed in Essay 15, cryptographic checkfunctions are formed by encrypting the variable-length source message, discarding all of the ciphertext except for a fixed-length residue, and using this residue as the checkfunction.

A variation of the cryptographic checkfunction is the use of a public key digital signature mechanism to securely sign a variable-length message or document. The resulting public key digital signature is fixed length and is appended to the unencrypted electronic image of the

document. Using this cryptographic method, a sender can assure a recipient of the document and digital signature that the source and integrityof the message are maintained.

For example, a hash function algorithm can be used to reduce the variable-length message to a fixed size, such as a message digest of 128 bits. The resulting mathematically unique hash value or message digest can be operated on by the sender's private key, such as the private key in RSA (Rivest, Shamir, and Adleman), to produce a digital signature. A digital signature provides integrity protection against viruses as well as other forms of unauthorized modification. A public key digital signature is the cryptographic result of the sender's private key operating on the message digest. With a public key algorithm, such as RSA, this process is called signing the message. The digital signature can be attached to the electronic image of the message and both encrypted using the recipient's public key. Only the recipient can decrypt the resulting message with his or her private key. After the recipient decrypts the digital signature with the sender's public key, the recipient has assurance that the signed message could only have come from the sender.

In the preceding example of a digital signature, authentication of the sender and message confidentiality can be provided together. The example discussed above is based on providing these combined security services by encrypting the message and its corresponding digital signature in the recipient's public key. In practice, high-volume encryption is generally achieved with more complex procedures that may involve one algorithm for key management or exchange and another for message confidentiality or privacy. Some of these issues are discussed in Essay 15.

Most malicious code operates by modifying other code or data. Modification controls are, therefore, appropriate countermeasures. Until the late 1980s, most attention in information security was focused on confidentiality. Some work was done on modification controls. Three approaches are discussed briefly below. These approaches are discussed in detail in Essay 8. Biba strict integrity provides protection by imposing hierarchical integrity levels and nonhierarchical integrity categories that govern which subjects may modify which objects. Biba strict integrity could be used to protect certain program and data files by marking them at the highest level or in a unique category. Clark-Wilson verified transactions can be used to control which programs, as well as which users, can modify which files. For example, only specified systems programs, using specified development tools, can modify operational programs. Type enforcement in lock extends the idea of verified transactions by specifying the types of programs that can modify files in specified domains.

Undesirable side effects of controls. Most of the countermeasures tend to limit information sharing. Systems that previously practiced open access are impacted.

Unfortunately, the misconduct of a few people can spoil the environment for the many who are well behaved. Some people believe that the imposition of controls can impair creativity.

Technical security, which primarily consists of confidentiality and integrity controls, generally has to be incorporated into life-cycle development of an information system. In any case, this is a fiscal and operational burden. It may be prohibitively difficult to add certain types of security measures to existing systems. Many countermeasures are considered as research topics. Operational overhead may be increased by technical countermeasures. Security operations may be user unfriendly.

However, when EDI is implemented for electronic commerce, there is generally an overall cost reduction to the purchaser of goods. Therefore, in those EDI cases where there are lower overall costs than with the predecessor information system and purchaser inventory maintenance operations, technical security costs are masked by the overall reduction in operational costs. For example, if an organization that manufactures automobiles uses EDI and related network operations with its paint suppliers to support just-in-time assembly of vehicles, the resulting information system and paint system inventory costs will drop considerably.

In this case, the EDI technical security costs may include the use of a MAC (message authentication code) for message integrity using DES (Data Encryption Standard) and a public key management system. However, the overall operational and paint inventory costs for the automobile manufacturer will tend to be less than for paper-based (first-class mail) purchase operations and larger on-site paint inventories.

Therefore, certain technical security methods to support electronic commerce may be associated with overall cost reductions. In addition, since EDI involves the development of new systems and security standards, there are opportunities to transfer much of the burden of access control and network security to cryptographic-based information-processing systems.

Legal remedies. Since malicious software has caused damage or at least inconvenience, it is natural to ask what protection is available under law. You may think of civil remedies, such as recovering damages by suing, or criminal punishment for the malefactor. The legal aspects are at least as dynamic as the technical ones. Enactment of legislation and its application have to lag technical changes in threats until the situation is well enough understood. Furthermore, the applicable law varies among jurisdictions. We can only generalize here.

Can hackers be sued for damages? There are legal theory problems. There may be no clearly applicable law. Extrapolation of existing law — such as trespass, conversion (recovery of damages caused by theft), and negligence — is possible but is subject to judicial restraint. New legislation is slow in coming, in part because it is difficult to write good laws. There are also practical problems. The cost of suing can exceed recovery from a hacker defendant. There are proof problems, such as a lack of paper records and witnesses. Cause and effect may be difficult to establish. For example, it may be difficult for the prosecutor to demonstrate that a hacker intentionally committed the act and without authorization [GEMI89].

A similar situation exists with respect to criminal law. Under British and US jurisprudence, a crime is an act declared to be illegal in a duly enacted statute. The law must be clear enough to give reasonable notice of the prohibited act. Ambiguity in drafting or error in specifying elements can preclude prosecution. The law must also specify the punishment. Even if there is an apparent violation of the law, a prosecutor may choose not to press charges. In the face of limited resources, prosecutors may exercise judgment concerning the cases which present the greatest danger to society. Lack of computer understanding on the part of prosecutors, judges, and jurors may also factor into the prosecutor's decision. There are also problems concerning the sophisticated circumstantial evidence usually required in computer crime prosecutions.

Notwithstanding these legal challenges, there are international trends to reconsider and strengthen a wide variety of laws pertaining to computer crime. Several European nations, such as Germany, France, and the United Kingdom, have already amended some of their laws pertaining to computer crime. In addition, the Netherlands recently reconsidered the need for legislation to restrict computer crime or "computer peace disturbance." One issue considered in the Netherlands proposals is the security of the attacked computer system. In other words, a conviction for a computer crime may be dependent on the existence of a "clear threshold" of security controls in the attacked system.